



DIMOCK STRATTON CLARIZIO LLP
Barristers and Solicitors • Patent and Trade-mark Agents

MARK B. EISEN
Ext. 242
meisen@dimock.com

Certified by the Law Society as a Specialist
in Intellectual Property (Patent) Law

SENT BY COURIER

September 4, 2003

Commissioner for Patents
U.S. Patent and Trademark Office
2011 South Clark Place
Crystal Plaza 2, Lobby, Room 1B03
Arlington, VA 22202
U.S.A.

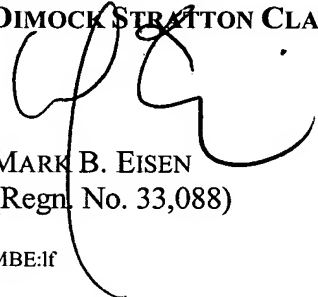
Dear Sir:

Re: Our File: United States Patent Application No. 10/644,818
Filing Date: August 21, 2003
Title: DIGITAL VIDEO SECURITY SYSTEM
Applicant: Strategic Vista International Inc.
Our File: 894-14/JLW

We enclose herewith a certified copy of the priority application, namely Canadian Patent Application No. 2,399,269 filed August 21, 2002, for filing in this application.

Yours very truly,

DIMOCK STRATTON CLARIZIO LLP


MARK B. EISEN
(Regn. No. 33,088)

MBE:lf

Encl. certified copy of CA 2,399,269



Office de la propriété
intellectuelle
du Canada

Un organisme
d'Industrie Canada

Canadian
Intellectual Property
Office

An Agency of
Industry Canada

*Bureau canadien
des brevets*
Certification

La présente atteste que les documents
ci-joints, dont la liste figure ci-dessous,
sont des copies authentiques des docu-
ments déposés au Bureau des brevets.

*Canadian Patent
Office*
Certification

This is to certify that the documents
attached hereto and identified below are
true copies of the documents on file in
the Patent Office.

Specification as originally filed, with Application for Patent Serial No: **2,399,269**, on
August 21, 2002, by **LARRY KLEIN AND JOEL KLIGMAN**, for "Digital Video
Security System".

Brady Pouches
Agent certificateur/Certifying Officer

August 25, 2003

Date

Canada




(CIPO 68)
04-09-02








OPIC  CIPO

DIGITAL VIDEO SECURITY SYSTEM

This invention relates to a digital video security system, as described in the following User Manual and the source code attached as Schedule "A".

Table of Contents

1	<i>Read Me First</i>	1
1.1	Steps to Get Going	1
1.2	How to Use This Manual	1
1.2.1	User's Manual Layout	1
1.2.2	Symbols Used in This Manual.....	2
1.3	System Contents	2
1.4	Minimum System Requirements	3
1.5	Windows Help File	3
1.6	Support	3
1.7	FCC Compliance	3
2	<i>Installing Digital Video Security System</i>	5
2.1	Installing the Software	5
2.2	Connecting the Hardware	8
2.3	Setting Up/ Installing Your Video security Camera.....	8
2.4	To Uninstall	9
3	<i>Tutorial</i>	11
3.1	Launching the Application	11
3.2	Quick Overview of the Interface.....	11
3.2.1	The Main Application Window: An overview	11
3.3	Getting Started	12
3.3.1	Setting up a Local Surveillance Connection 	12
3.3.2	Opening the Local Video Surveillance Connection 	15
3.3.3	Viewing Video Locally 	17

3.3.4	Adding Alarms 	18
3.3.5	Setting Actions in Response to an Alarm	20
3.3.6	Adding Scheduled Events 	27
3.3.7	Enabling Local Surveillance to be Viewed Remotely	33
3.3.8	Setting up a Remote Surveillance Connection	34
3.3.9	Viewing Live Video Remotely	37
3.3.10	Video Playback 	40
4	Reference	43
4.1	The Digital Video Security System Interface	43
4.1.1	The Menu Bar and Toolbar	43
4.2	Yellow Pages Directory	51
4.2.1	Registering a Host with the Yellow Pages Directory 	52
4.2.2	Unregister a Host from the Yellow Pages Directory 	54
4.3	Setting Up Connections	54
4.3.1	The Connections Window	55
4.3.2	Setting Up a New Local (Host) Connection	59
4.3.3	Setting Up a New Remote Connection	61
4.4	The Local Surveillance Window	66
4.5	Motion Detection 	72
4.6	Remote Surveillance Window	75
4.7	Scheduling Alarms	80
4.8	Scheduling Events	82
4.9	Actions	83
4.9.1	Action: Sending E-mail	86
4.9.2	Action: Dial a Phone or Pager	87
4.9.3	Action: Record Video	89
4.9.4	Action: Control X-10 Devices	89
4.10	Video Playback 	91
4.11	Feedback	98
4.12	Configuration	100
5	X-10 Background Information	103
5.1	An Overview of the X-10 Protocol	103
5.2	Using X-10 With Digital Video Security System	103
5.3	Transmitting X-10 Commands	104
6	Data Maintained by Digital Video Security System	105
7	Glossary	107

Chapter 1: Read Me First

1 Read Me First

The Digital Video Security System enables you to monitor and protect your business, home or office from across the hall or around the world. The video grabber software provided with Sylvania's Digital Video Security System, allows you to set up a PC based security system in minutes that can:

- **Stream Live Full-Motion Video** – Watch what is going on in the next room, or around the world via the internet or dial up option
- **Detect Motion** – When the alarm feature is activated, the system will automatically dial a telephone or pager number that you have selected and play a customized message. You can also have the system send video email.
- **Record Digital Video** – Manage digital recording sessions or save the video as a streaming file that can be replayed locally or e-mailed to a remote location. Review the video using still frame, forward, fast forward and other video management capabilities.
- **Control Home Automation** – Use any X10 Home Automation and Wireless devices to control lighting and other features available with X10-compatible systems
- **Schedule Events** – Schedule video recording, e-mail notification, X10 Home Automation and more based on a user-defined schedule

1.1 Steps to Get Going

Listed below are the recommended steps for installing Digital Video Security System:

- Check your package using the list on the next page to make sure you received the complete system.
- Confirm that your computer meets the system requirements for Digital Video Security System.
- Install Digital Video Security System. You can install the application at once or choose the ones you need at this time.
- Register as a Digital Video Security System user.

1.2 How to Use This Manual

This user's manual describes the contents of Digital Video Security System and system requirements. It also provides installation instructions and tells you where to go for more information.

1.2.1 User's Manual Layout

This manual is designed to get you started quickly, while providing you with a full reference. Each chapter is self-contained so that the manual does not have to be read in sequence.

The manual is divided into the following chapters:

Chapter	Title	Description
2	Installation	Describes installation and setup of hardware and software
3	Tutorial	Overview of the main software features
4	Reference	Full feature description of the each dialog
5	Appendix 1	General Overview of the X-10 protocol
6	Glossary	Definitions of common terms used in this manual
7	Index	Index lookup

Chapter 1: Read Me First

Screen shots in this manual have been taken on a system running Windows XP. There will be small differences to the external appearance of the application windows and dialogs running on Windows 2K. The layout of the windows and dialogs and their content are identical for all Windows systems supported.

1.2.2 Symbols Used in This Manual

Throughout this manual a set of comments are employed to provide emphasis to certain points. A left-hand icon indicates the type of comment as follows:



This type of comment represents a feature or aspect of the Digital Video Security System that is particularly beneficial to the user. Text in this note is italicized and bold.



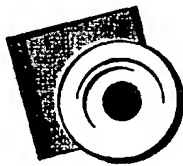
This type of comment represents information that you will find useful, such as a shortcut or a "how-to" to avoid common mistakes. Text in this note is bold.



THIS TYPE OF COMMENT REPRESENTS SOMETHING TO BE AWARE OF OR BE CAUTIOUS ABOUT. TEXT IS BOLD AND UPPERCASE.

1.3 System Contents

- **Video Grabber** – A PC interface that allows you to connect a video device, such as a security camera, to your PC.
- **Software** – Digital Video Security System Application software to enable you to monitor and protect your business, home or office from across the hall or around the world.
- **USB Cable** – Used to connect Video Grabber to your computer's USB serial port.
- **Color Video Security Camera (included with VG1100 system)** – A high-quality color video security camera with cable and 12 V DC power supply. Note: if you purchased VG1000, you will need to purchase a video security camera.
- **User's Manual** – Full reference to the features of Digital Video Security System.



Video Grabber

USB Cable

Application
Software CD

Owners Manual

Color Video
Camera
(Included with
Model VG1100)

Chapter 1: Read Me First

1.4 Minimum System Requirements

The minimum system requirements for the Digital Video Security System are:

Processor	Pentium II or higher
RAM	32 MB Ram
Windows O/S	Windows 98, Windows ME, Windows 2000 , Windows XP
Video Card	
Display	800 x 600 VGA
Sound Card	
Disk Space	40 MB available hard disk space
Modem	28 bps or higher modem
CD-ROM Drive	Required for software installation
Computer Connection	USB
Camera	Included with VG1100 (not included with VG1000)

1.5 Windows Help File

The Digital Video Security System User's Manual is available in modified form as a Windows Help file. To access the online Help file, select Help from the Digital Video Security System menu off the Start | Programs menu, or from within the Digital Video Security System application itself.

1.6 Support

The following 3 options are available for technical support:

Type	Contact	Hours
On-line Support	Please visit our website at www.strategicvista.com for free technical assistance anytime	24 Hours / 7 days a week
Telephone Support	Should you need to talk with a customer support representative please call 617-746-2982. Please note that a fee may apply for this service.	
Email	Please email your queries to us at info@strategicvista.com	

1.7 FCC Compliance

This equipment has been tested and found to comply with the limits for a class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection



Chapter 1: Read Me First

against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communication. However there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna increases the separation between the equipment and receiver
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- Consult the dealer or an experienced radio or television technician for help.

This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. This device must accept any interference received, including interference that may cause undesired operation

This digital apparatus does not exceed the Class B limits for radio noise emissions from digital apparatus set out in the Radio Interference Regulations of the Canadian Department of Communications.

Chapter 2: Installing Digital Video Security System

2 Installing Digital Video Security System

This chapter describes how to install Digital Video Security System. You will need a video security camera (included with VG1100) to operate this system.



DO NOT PLUG THE VIDEO GRABBER INTO THE USB PORT OF YOUR COMPUTER. YOU MUST FIRST RUN THE INSTALLATION SOFTWARE.

2.1 Installing the Software

To install the Digital Video Security System software application, simply insert the CD into your computer's CD-ROM. After a few seconds the Setup application will automatically run.

On some systems the Windows auto-start is turned off. If the Setup application does not run within a few seconds after you place the installation CD in the CD-ROM drive, go to the Windows Explorer and run Setup from the CD-ROM drive by double-clicking the Setup.exe file.

When the Setup starts up you will see the Windows Installer main window:

Follow the instructions of the standard Windows installation dialog, which will guide you through the installation process.



YOU MUST ACCEPT THE LICENSE AGREEMENT PANEL OR THE INSTALLATION WILL NOT CONTINUE.

The next two panels will require you to accept the license agreement, and enter your customer information. You will then be prompted with the following Setup Type panel:



Chapter 2: Installing Digital Video Security System

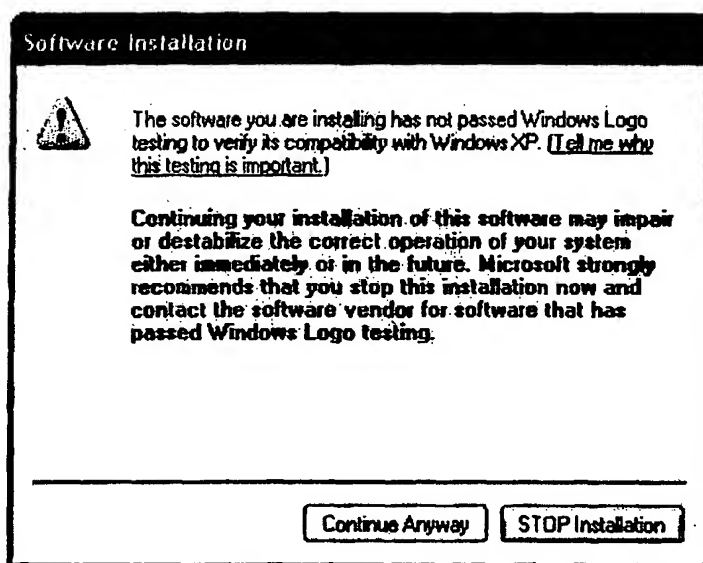
Select the Complete Setup option and click on Next. The installation may appear to pause during the installation process. If this happens let the installation software proceed. It is automatically configuring your Windows environment to determine what it needs to install in order for the Digital Video Security System software application to operate properly. The window shown below will update you as to the progress of the installation:

The installation automatically installs the video camera drivers, which include a video camera as well as an audio driver (the audio capture device is built into the camera for the VG1100). You will see the installation launch the following screen that indicates the video camera drivers are about to be installed:

Chapter 2: Installing Digital Video Security System

Simply proceed with the installation by clicking the Next button.

On Windows XP you will see the following window during the video driver installation:



Simply press Continue to proceed. On Windows 2000 a variation of this window appears when you install the Video Grabber into the USB port. Simply press Continue as well.

When the installation is completed you will see the following window:

Chapter 2: Installing Digital Video Security System

Depending on what the Setup application needed to install, you may be prompted to restart Windows in order to complete the installation.



SINCE DRIVERS ARE INSTALLED IT IS RECOMMENDED TO RESTART YOUR SYSTEM EVEN IF THE INSTALLATION DOES NOT REQUIRE IT PRIOR TO PLUGGING IN YOUR VIDEO CAMERA.

2.2 Connecting the Hardware

Once the installation is completed connect the Video Grabber to the USB port on your computer using the supplied cable.



The Video Grabber is designed as a Plug-and-Play device. The Windows device manager will automatically find the Video Grabber and install the correct video drivers.

You can now plug any video security camera into the Video Grabber (included with VG1100) and start monitoring your business, home or office.

2.3 Setting Up/ Installing Your Video security Camera

For users who purchased model VG1000, refer to your Video Security Cameras owners' manual for installation of your security camera. Once the security camera has been set up, simply connect the RCA cable from the camera to Video Grabber.

For users who purchased model VG1100, proceed with the following steps:

Digital Video Security System User's Manual

Chapter 2: Installing Digital Video Security System

1. Attach the camera bracket to the camera. Position the security camera in the desired viewing location. **Note:** A color camera requires a certain level of light. You may need to adjust the camera position and/or the amount of light in the room for optimum viewing.
2. Run the supplied 17-meter (57ft) cable from the camera to video grabber. Connect the RCA leads to Video Grabber and plug the camera into an electrical outlet using the supplied 12 V DC power supply.

Refer to Technical Specifications at the back of this manual for detailed information on the camera.



If this was your first time setting up a security system, congratulations! You just set up a sophisticated digital video monitoring system in minutes! Continue to the next chapter and you will be watching video in another couple of minutes.

Once the camera is plugged in Windows Plug-and-Play will prompt you to confirm that the drivers are installed. You will be prompted with the following dialogs:

Simply press the OK button to accept the drivers.

2.4 To Uninstall

To uninstall Digital Video Security System go to the Add/Remove Programs option in the Control Panel of Windows. First click on the DVS System item to uninstall the main Digital Video Security System application. The Digital Video Security System Uninstall program will prompt you with the following window:

Select Yes and follow the on-screen instructions to remove the Digital Video Security System application.

After the Digital Video Security System is uninstalled you must separately uninstall the video camera drivers by selecting the Video Capture item in the Add/Remove Programs list. You will be prompted with the following window:

Chapter 2: Installing Digital Video Security System

Simply click OK and continue with the on-line instructions.



YOU MUST REMOVE THE VIDEO CAPTURE DRIVERS SEPARATELY AS DESCRIBED ABOVE BEFORE REINSTALLING THE DIGITAL VIDEO SECURITY SYSTEM OR YOU MAY RUN INTO CONFLICTING DRIVER ISSUES.

3 Tutorial

This tutorial will introduce you to the basic concepts of the Digital Video Security System application, with the goal of getting you up and running in 15 minutes. It will provide you with an understanding of how to navigate through the Digital Video Security System software to set up a basic security system with a minimum of effort. A more complete description of all of the powerful features of the Digital Video Security System application is contained in the Reference, Section 4.

This tutorial uses a case study approach that assumes you are the owner of a thriving store. Business is great and you have just hired a new employee to manage the store in your absence. You are working out of your home office and would like to do the following:

- Monitor how the cash register is closed down for the day;
- Check on your store at random during the day from your home office;
- Ensure that there are no intruders after business hours.

You can use the Digital Video Security System software for all these capabilities, and more. Let's get started!

3.1 *Launching the Application*

To launch the Digital Video Security System application, select it from the Programs | Digital Video Security System menu option, or click on the Digital Video Security System icon in the Digital Video Security System folder.

3.2 *Quick Overview of the Interface*

Digital Video Security System is comprised of a main application window from which you launch and control the specific features of the application

3.2.1 **The Main Application Window: An overview**

The main window contains four areas and appears as follows:

The menu and icon bars provide you with access to the feature set of the Digital Video Security System application. The main window is where all launched windows will open, and the status bar provides you with feedback on any operations Digital Video Security System is carrying out. By default, a Connections window and a Feedback window will be opened in the main Digital Video Security System window.


Chapter 3: Tutorial

3.3 Getting Started

Using the case study scenario described in the introduction to the tutorial, your store will be set up as the local surveillance connection. The local surveillance is always the location of the video camera. Later in this tutorial you will set up your home office as the remote surveillance connection. The remote surveillance is the location from which you will remotely monitor and control the security at the store.

3.3.1 Setting up a Local Surveillance Connection

On the main menu icon bar, click on the Connections icon, or select View | Connections on the main menu. This will open the Connections window in the main window, as shown below:



Chapter 3: Tutorial

You will use this window to set up the local connection to the video camera in your store.




A local connection is set up at the computer that has the Video Grabber, camera and Digital Video Security System software installed. It is referred to as Local Video because the security camera is local to this computer. The Digital Video Security System will stream video from this location to the internet or directly to another computer, hence it is sometimes also referred to as the host computer.



Note that the Connections Window has its own toolbar. All windows have their own toolbar for added convenience and ease-of-use.

Select the
presented with the following Add Connection dialog box:

New Connection icon. You will be



Chapter 3: Tutorial

Type a descriptive name in the label field; in this example "Store Surveillance Camera".

The Connection Information area will have Video Surveillance preselected as the Type of Connection. From the drop-down Connection Method Menu, select Local (TCP/IP Host)

Specify your video and audio devices by simply selecting the desired device from the drop-down menu.

Login information is optional (refer to Section 4.3); for this tutorial leave these fields blank.

Select OK, and the new connection is added. It is that simple! You will see your new connection listed in the Connections window, as shown below:



All connections are automatically saved when you exit the application. You do not have to specifically save your connections.

3.3.2 Opening the Local Video Surveillance Connection

Now that the connection has been set up, you will open the connection window in order to:

- Monitor the video camera locally
- Enable *remote* locations to monitor the local camera
- Optionally establish alarms and set up the corresponding trigger actions
- Optionally set up scheduled events

To open the connection you just set up, select the Store Surveillance Camera connection by either double-clicking on its label, or by highlighting the label with your mouse and selecting the

Open Connection icon from the Connection window. The following Local Video Surveillance window will open:

Chapter 3: Tutorial




The Local Video Surveillance maintains the connection label in the title bar to identify which connection it is controlling. This is the descriptive label that you entered when creating the connection.

Chapter 3: Tutorial

3.3.3 Viewing Video Locally

You can instantly view video locally by pressing the

Play icon on the Local Video Surveillance toolbar. This lets you preview what will be transmitted remotely, and enables you to adjust your camera appropriately for both local and remote viewing.



Chapter 3: Tutorial

To stop the video camera from displaying in the window, simply press the

Stop button on the toolbar.

3.3.4 Adding Alarms

In this next section, we will cover how to set up an alarm that will detect security breaches based on motion detection. An alarm can trigger the following events:

- send a notification via e-mail
- automatically dial a phone, pager or PDA
- capture the motion on video
- turn ON or OFF lights or other electrical devices using X-10 controllers

Chapter 3: Tutorial

In order to set up an alarm for Motion Detection, click on the

New Alarm icon in the Local Video Surveillance window. The Motion Detection Alarm dialog will open, as shown below


:

Type in a descriptive label for the alarm; in this example, "Weeknights".



You can set up multiple alarms, Each one is identified by a unique label that you select. Refer to the reference section for more details.

Fill in the Begin and End times that the Motion Detection Alarm should be active. For example, in this case we will select the nighttime hours when the store is closed. To change the times, position your mouse in the hours, minutes or AM/PM field, and use the up or down arrows to scroll to your desired settings. Alternatively, you can highlight any of the fields and type the desired information in manually, one field at a time.



Chapter 3: Tutorial

In the Scheduled Days region, select the  button.
This will automatically check the days from Monday through Friday.



If you accidentally pressed OK and closed out of this dialog, your information is still saved. Click on the Alarm label in the Local Video Surveillance window to re-open the Motion Detection Alarm dialog.

The Motion Detection Alarm dialog will now appear as follows:

In the next section, you will add the **Actions** that should occur upon detection of motion such as sending an e-mail (section 3.3.5.1), dialing a phone or pager (section 3.3.5.2), recording a video (section 3.3.5.3) or activating an X-10 devices (section 3.3.5.4).

3.3.5 Setting Actions in Response to an Alarm

You can set up as many actions as you want as a result of motion being detected, including multiple occurrences of the same type of action. This section of the tutorial reviews setting up one of each possible action.

Chapter 3: Tutorial

3.3.5.1 Send an E-Mail

To add an action that sends e-mail as a result of motion detection, select **Send e-mail** from the Add Action drop-down box in the Motion Detection Alarm window and click Add. You will see the following dialog:

As with any e-mail system, enter the address and fill in the subject and message fields.


Before selecting OK, you can first test the e-mail by selecting the Test option. This will immediately send the e-mail to the selected address, so you can ensure it will be sent to the correct recipient in the case of an alarm.



All of the Action dialogs give you the option of testing your settings. Testing your settings confirms that your e-mail (or any other action) ends up where you expected.

Select OK. With these simple few steps, you have established that if motion is detected between the hours of 9:00 PM and 8:00 AM on weekdays, a prespecified text e-mail alert will be sent. You also can send a video attachment along with the email; see Section 4.9.1 for more details.

The Motion Detection Alarm window will appear as follows:



Chapter 3: Tutorial



In the window that lists the actions, the Details column provides a brief summary of the action so that you can easily identify it.

3.3.5.2 Dial a Phone or Pager

To add an action that places an automated call to a phone or pager as a result of motion detection, select the **Dial Phone or Pager** option from the drop down Action Menu and click ADD. You will see the following dialog:

Enter the phone number of the emergency contact and the dial (DTMF) tones you will be sending to the pager. (For full details on the various aspects of this Action, please refer to Section 4.9.2). You can optionally place dashes ('-') to present the phone numbers in familiar form (999-9999), though this will not affect how the number is dialed out.



A DTMF Tone is simply the sound you hear when you press the buttons of a standard phone. DTMF Tones correspond to the numbers on the phone keypad (i.e., 0 through 9).

To ensure that the call goes where you intended, use the Test feature. This will place an immediate call according to the information you have entered.

Chapter 3: Tutorial



Sending a test phone pager message does not affect any protection when motion is detected. It is merely a convenience to ensure that you entered information correctly into the dialog.

Press OK. Now you have established that, in the event of motion detection during the hours of 9:00 PM and 8:00 AM during the week, two actions will take place: a prespecified e-mail will be sent and an automated call will be placed to a pager. The Motion Detection Alarm window will appear as follows:

Note that two Actions to the same alarm are now listed in the Actions region of the Motion Detection Alarm window.

3.3.5.3 Record Video

It would also be convenient, for security purposes, to capture on video the cause of the motion detection. Let us establish that, if motion is detected, Digital Video Security System will automatically begin recording a 2-minute video.

From the Motion Detection Alarm dialog, choose **Record video** from the Add Action drop-down list and click Add. The following dialog will appear:

Chapter 3: Tutorial

Type in a descriptive title for the video capture (e.g., front door) and select 2 minutes as the duration by highlighting the minutes field and using the up and down arrows at the right of the duration box.



DIGITAL VIDEO RECORDING OCCUPIES A LOT OF DISK SPACE AND SHOULD THEREFORE BE USED CAUTIOUSLY, ESPECIALLY ON SYSTEMS THAT DO NOT HAVE A LOT OF FREE DISK SPACE. EACH MINUTE OF VIDEO RECORDING IS APPROXIMATELY 1.5 MB.

Make sure that the *Use default video record file* box is checked.



It is generally better to use the default video record file, which automatically names the file based on the date and time and will appear in the Video Playback window automatically.

You can run a test immediately by pressing the Test button to record video for the pre-specified duration of 2 minutes. The system will store the video capture in the default video record file. (For more on default video record files, please refer to Section 4.12).

Press OK. You have now established that, in the event of motion detection during the hours of 9:00 PM and 8:00 AM during the week, three actions will take place:

- a pre-specified e-mail will be sent;
- an automated call will be placed to a pager;
- 2 minutes of video will be captured and stored for future reference.



If you would like the video recording to be attached to the e-mail, switch the order and have the video recording set as the first action prior to the e-mail action. All actions are executed in the sequential order in which they are listed.

The Motion Detection Alarm window will appear as follows:

Note that three Actions to the same alarm are now listed in the Actions window of the Motion Detection Alarm window.

Chapter 3: Tutorial



Actions are executed in sequential order. This typically is only relevant when you want to record video that will then be used as an e-mail attachment in a subsequent action.

3.3.5.4 Activate X-10

To add an action that controls X-10 devices such as lights and other electrical outlets as a result of motion detection, select **Control X-10 devices** from the Add Action drop-down box in the Motion Detection Alarm window and press Add. You will see the following dialog:



In order to use the X-10 capabilities, you need to purchase a CM17a X-10 controller, otherwise known by its nickname "FireCracker". The FireCracker device is a small connector that plugs into the Serial port of your computer and transmits an RF signal to an X-10 receiver that is plugged into the wall.

Enter the House and Device Codes that, in combination, might identify the lights in the front of the store. You may either input the code values using your keyboard, or select the desired value from the drop down boxes to the right of each field. (For details about valid input values for these fields, refer to Section 4.9.4; for a brief overview of the X-10 protocol, please see Section 5.)


Set the Command to ON, indicating that the lights should turn on when motion is detected.

Running Test will turn on the selected lights immediately, confirming that you have accurately entered the Codes you desired.

Press OK. You have now established that, in the event of motion detection during the hours of 9:00 PM and 8:00 AM during the week, four actions will take place:

- a prespecified e-mail will be sent;
- an automated call will be placed to a pager
- 2 minutes of video will be captured and stored for future reference
- lights in the front of the store will turn on.

The Motion Detection Alarm dialog will appear as follows:



Chapter 3: Tutorial

Note that four Actions to the same alarm are now listed in the Actions region of the Motion Detection Alarm window.

You may close the Motion Detection Alarm dialog by pressing OK. Your Local Video Surveillance window will now reflect the Alarm you have just added:



You can edit the Alarm you just created by double-clicking on the Weeknights alarm icon.

3.3.6 Adding Scheduled Events

Let us now focus on the events that you may choose to schedule on a regular basis. A Scheduled Event is a set of actions that are designated to occur on a weekly basis at pre-designated times. One example might be to have all the lights automatically turn off at night, or recording the closing out of the cash register every evening. Adding a Scheduled Event is as easy as adding an alarm and can include:

- Sending an e-mail
- Automatically dialing a phone or pager
- Recording video
- Controlling X-10 devices



The Scheduled Event dialog enables you to flexibly select events that can be scheduled in any combination of days or times of the day, such as weeknights, weekdays, weekends, etc.

To demonstrate the simplicity of adding a Scheduled Event, which is nearly identical to adding an Alarm, let us walk through the two examples mentioned above, which are setting the lighting at a pre-designated time and recording the cash register every evening.

Chapter 3: Tutorial

3.3.6.1 *Open Scheduled Event Dialog*

From the Local Video Surveillance window which should still be open, click on the

appear:

Add Scheduled Event. The following dialog will




Notice that the Schedule Event and Motion Detection Alarm windows are almost identical. You can apply all that you learned in the last section to quickly go through setting up actions based on a scheduled event.

With the exception of the Scheduled Time field, this dialog is identical to the Motion Detection Alarm dialog. (In setting up an Alarm, you choose both a start and end time; for a Scheduled Event you need to select only a start time.)

Type in a descriptive label; ours reads "Turn Off Lights."

Enter 10:00 PM in the Scheduled Time field by positioning your mouse in the hours, minutes or AM/PM field, and using the up or down arrows to scroll to the correct time. Alternatively, you can highlight any of the time fields and manually type in the desired information, one field at a time.



Chapter 3: Tutorial

In the Scheduled Days region, select the
This will automatically check all seven days.

button.

The Scheduled Event window will appear as follows:

3.3.6.2 Activate X-10

You can now specify which action you would like to schedule; namely, accessing the X-10 protocol to turn off all lights.



Setting up any of the Actions in a Scheduled Event – e-mail, dial a phone or pager, record video or activate X-10 – is identical to setting up an Action in response to an Alarm.

From the Add Actions drop-down list, select **Control X-10 devices** and press Add. The following dialog will open:

Chapter 3: Tutorial

Enter the House and Device Codes that, in combination, identify all of the lights in the store. You may either input the code values using your keyboard, or select the desired value from the drop down boxes to the right of each field. (For details about valid input values for these fields, refer to Section 4.9.4; for a brief overview of the X-10 protocol, please see Section 5.)

Set the Command to OFF, indicating that all lights should be shut off.

Running Test will turn off the selected lights immediately, confirming that you have accurately entered the Codes you desired.

Press OK.

The Scheduled Event window will appear as follows:

You have now established that every night, at 10:00 PM, all of the lights in the store will be turned off.

Press OK on the Scheduled Event Window to save these settings. As shown below, the Local Video Surveillance window will reflect that both an Alarm and a Scheduled Event have been established:



Chapter 3: Tutorial

3.3.6.3 Record Video

From the Local Video Surveillance window shown above, select the


Schedule Event icon. When the Schedule Event window opens, enter the following information:

Event Description: Cash Register Closing

Scheduled Time: 9:00 PM

Scheduled Days: Check all days but Sunday

Your Schedule Event window should look like this:



Chapter 3: Tutorial

From the Add Action drop-down list, select **Record video** and press Add. When the Video Record Action dialog opens, input the following information:

Title: Nightly Cash Closing

Duration: 5 minutes


Use default video record file: checked

The Video Record Action Dialog will look like this:

Running Test will immediately capture the prespecified 5 minutes of video, and will store the video capture in the default video record file. (For more on default files, please refer to Section 4.12).

Press OK. You have now established that six nights a week, at 9: 00 PM, Digital Video Security System will record a 5-minute segment of video and store it for future reference.

The Schedule Event window will appear as follows:



Chapter 3: Tutorial

Select OK to close the Schedule Event window.

Your Local Video Surveillance window will now reflect that you have established one Alarm and two Scheduled Events for the Store Surveillance Camera:

3.3.7 Enabling Local Surveillance to be Viewed Remotely

Once your local camera is set up properly in the store, you can enable it to be viewed remotely by others. When you open the Local Video Surveillance you can control whether a remote workstation can view the local video camera using the

Connect and

Disconnect icons found on the Local Video Surveillance toolbar. Once the Local Video Surveillance is connected, it is ready for a remote workstation to view the local video camera, as described in the next section.


3.3.8 Setting up a Remote Surveillance Connection

Once your local camera is properly set up in the store, you can move on to your home office, where you will set up a Remote Video Surveillance connection. Using this remote connection, you will be able to:

- **remotely** monitor the live video feed from the store's security camera
- **remotely** establish new Alarms or Scheduled Events for the store
- **remotely** modify existing Alarms and Events



There are two methods for establishing a remote connection to the host (local) computer: Fixed IP Address and Yellow Pages Entry. This tutorial demonstrates how to remotely connect to a host computer that has a Fixed IP Address. For details on remotely connecting to a host computer using the Yellow Pages Entry, please refer to Section 4.3.3.2.




Chapter 3: Tutorial

Install the software at your home office location following the directions in Section 2.1. Launch the application, as per instructions in Section 3.1.

From the main window of the Digital Video Security System application, click on the

Connections icon, or select View | Connections, as you did in Section 3.3.1. Once the Connections window has opened, select the

New Connection icon. Fill in the fields of the Add Connection dialog box as follows:
Type in a descriptive label for this remote connection: "Home Office".
The Type of Connection will be preselected as Video Surveillance.
Use the Connection Method drop-down list to select Fixed IP Address.
Your Add Connection dialog will now appear as follows:



Chapter 3: Tutorial

Please note that the Connection dialog now has a new field requesting the Fixed IP Address. Enter the IP address of the computer at the store where you have the camera connected. (For more details on Connection Methods for Remote Connections, please refer to Section 4.3.3.)

Press OK.

That's it! Your Remote Surveillance Connection has been set up!

As shown in the screenshot below, the Connections window displays all of your established connections.

Icons on the left represent whether the connection is local or remote, as follows:

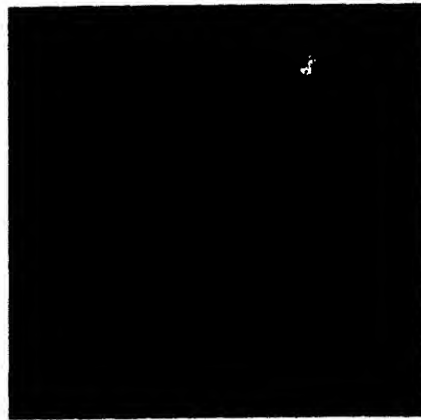
Connection Icons	Description
------------------	-------------

	Denotes a LOCAL connection (i.e., the camera is connected to the PC at this location)
	Denotes a REMOTE connection (i.e., will connect you to a video camera at a remote location)

From the **Connections** window (as shown in the previous section), select which remote video connection you would like to access by double-clicking on its label with your mouse, or

Chapter 3: Tutorial

highlighting the label and selecting the **Open** Connection icon. You will be presented with the Remote Video Surveillance window, as shown:




Chapter 3: Tutorial

To remotely view the security camera footage live, press the

Connect Icon. The video will play in the right side of the Remote Surveillance Window,

To stop the viewing video from the remote location, simply press the

Disconnect icon. You can connect and disconnect to the remote video as often as necessary. For further details about remotely viewing live video, please refer to Section 4.6




Chapter 3: Tutorial

3.3.10 Video Playback

To replay previously captured video, use Digital Video Security System's convenient Video Playback feature. From the main window, select the

Video Playback icon. The following window will
open:



Chapter 3: Tutorial

On the left-hand side of the window, you will see a listing of all video files stored in your default directory. (For more details on Configuring your default video directory, refer to section 4.12.) To view the video capture, double-click on the video of your choice. Video is played in the right-hand panel of the Video Playback window, using the embedded Window Media Player.



When the video listing is in Detail mode, it automatically lists the date, start and stop times, and Local Video Surveillance camera that recorded the video.

For more details on Video Playback and navigating the Windows Media Player, please refer to Section 4.10.

4 Reference

4.1 The Digital Video Security System Interface

The main Digital Video Security System application window is comprised of the following elements:

- the menu bar
- the toolbar
- the status bar
- the main window or work area

Upon launching Digital Video Security System, the default layout of the application will appear as follows:



By default, two windows are opened in the main window of the application: Connections (refer to Sections 3.3.1 and 4.3.1) and Feedback (refer to Section 4.11). Using the menu bar and toolbar, you can rearrange the layout of the application in the way that is most convenient for you. By default, the layout is designed to fit an 800 x 600 pixel screen. Whether you maintain the default layout, or modify it to one that better suits your needs, Digital Video Security System will maintain your layout between sessions. That is, if you shut down the application and relaunch it at a later time, the main application window will appear just as you last left it. This is referred to as persistence – all values, such as the layout, connection information, and configuration settings, persist between sessions.

4.1.1 The Menu Bar and Toolbar

Use the menu bar to control which windows are being viewed within the Digital Video Security System, as well as to customize the layout of the application window. As an alternative to the menu bar, the toolbar provides you with a convenient, single-click method of accessing main menu functions.

The menu bar consists of the following top-level menu items:

- File
- View
- Help

4.1.1.1 File

The File menu provides you with the following options:

Chapter 4: Reference

The table below describes the function of each of these menu items, and displays the parallel single-click icon which accesses the same functionality.

Toolbar Icon	Menu Bar Label	Description of Function
	Register with Yellow Pages	Register your local (host) computer with the online Yellow Pages. This simplifies remotely connecting to the host PC. For an explanation of the Yellow Pages directory, refer to Section 4.2; for details on registering the local computer, refer to Section 4.2.1.

Chapter 4: Reference

	Unregister with Yellow Pages	Remove your local (host) PC from the online Yellow Pages directory. Refer to Section 4.2.2 for details.
	Exit	Close the Digital Video Security System application.

Whether you exit the application using the File | Exit option or quit by pressing

the , Digital Video Security System will maintain the persistence of your layout – it will always appear just as you last left it.

4.1.1.2 View

The View menu provides access to the core functional features of Digital Video Security System:

Chapter 4: Reference

The table below describes the function of each of these menu items, and displays the parallel toolbar icon, where one exists.

Toolbar Icon	Menu Bar Label	Description of Function
	Toolbar	Shows or hides the Toolbar. If the Toolbar is shown then there is a check mark next to this menu item.
	Status Bar	Shows or hides the Status Bar. If the Status Bar is shown then there is a check mark next to this menu item.
	Connections	Opens the Connections window in the main work area. Use this window to establish your link, whether local or remote, to the surveillance camera. There is a check mark next to this item if a Connections window is open. (Section 4.3)

Chapter 4: Referenc

	Feedback	Opens the Feedback window in the main work area. This window provides you with instant feedback on any actions carried out by Digital Video Security System. There is a check mark next to this item if a Feedback window is open. (Section 4.11)
	Video Playback	Opens the Video Playback window in the main work area. In this window, you can view previously captured video footage. There is a check mark next to this item if a Video Playback window is open. (Section 4.10)
	Configuration	Will open the Configuration dialog in the main work area. Use this dialog to set up your preferences and default file settings. (Section 4.12)

In addition, the area below the Configuration item displays a numbered list of all windows currently open in the Digital Video Security System application. There is a check mark next to the window that is currently active.

Selecting an alternate item from the numbered list will cause the newly selected window to be active by bringing that window to the forefront of the work area.

Chapter 4: Reference

4.1.1.3 Help

The Help menu provides you with access to the online Help file and support options:

Toolbar Icon	Menu Bar Label	Description of Function
	E-Mail Support	In order to facilitate communication with technical support, you can directly e-mail your Digital Video Security System settings. Selecting one of the three options – Configuration, Connections or Layout – will automatically open an e-mail dialog with the selected settings file as an attachment. You simply specify the address, add any explanatory comments, and press Send.

Chapter 4: Referenc

	About Sylvania Digital Video System	Provides you with important user information, such as which version of Digital Video Security System you are running.
--	---	---



Sending your tech support issues using the E-Mail Support facility will greatly enhance the ability to solve your tech support problem. The information that is contained in the attached files provides our tech support with a snapshot of your SDVS system, and can greatly reduce response time.

4.1.1.4 Status bar:

The status bar notifies you of key pieces of information regarding the current session of the Digital Video Security System. It displays the "tool tips" for any icon your mouse passes over, and notifies you of errors or other information that changes based on context.

4.1.1.5 View List Icons

Throughout the application, many of the windows allow you to customize the way information is listed. The following table describes the various view list icons and their functions:

Chapter 4: Reference

Icon	Description of Function
	Displays large icons
	Displays small icons arranged side by side

Chapter 4: Reference

	Displays small icons in a list format
	Displays small icons along with relevant details of the properties of the items

4.2 Yellow Pages Directory

The Yellow Pages Directory service is a free service maintained by Strategic Vista, the distributors of the Digital Video Security System. The Yellow Pages Directory works like a typical directory – by maintaining a listing of users and their IP addresses so that they that can be looked up in real-time. The purpose of the Yellow Pages facility is to enable remote Digital Video Security System users to look up host systems without having to remember their IP address. The Yellow Pages facility is particularly useful for Digital Video Security System local host computers that are linked to the Internet using dynamic IP addresses – addresses that change periodically when the computer logs onto the Internet. The Yellow Pages facility automatically determines the correct IP address of the local host and registers it with the Yellow Pages directory.

In the Digital Video Security System, the local host computer registers itself with the Yellow Pages Directory. Then, as an option, every time the local computer is started Digital Video Security System finds its IP address and updates the Yellow Pages directory. Thus, the Yellow Pages directory is continuously updated with the most current IP address for the local host.

Chapter 4: Reference

Registering the local computer with the yellow pages directory means that DVSS Users from remote locations can readily look up and locate your IP address. It is the simplest way to gain access to local host that has a dynamic IP address - many ISPs, DSL, and cable services do not provide a fixed IP address; or if a remote user does not know your IP address.




Even if your ISP, DSL or cable service does not provide fixed IP addresses, use the Yellow Pages Directory to connect to your Digital Video Security System host computer.



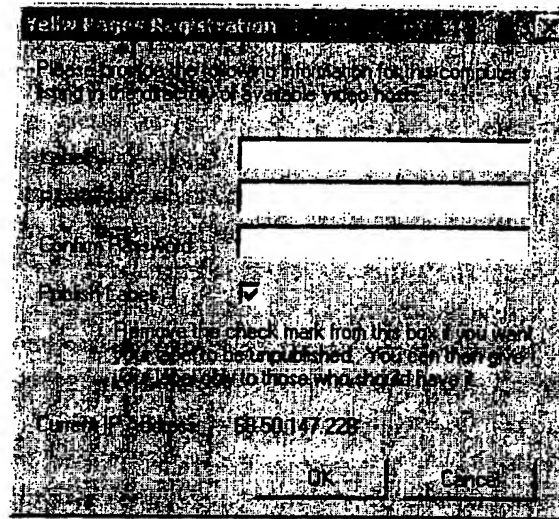
Security is maintained when a local host is registered in the Yellow Pages Directory in two ways: 1) The local host can be registered anonymously so that only a remote computer that knows the exact Yellow Pages Directory entry can find the local host, and 2) Remote users still need a valid Username and Password to enter the local host.

In order for a Yellow Pages Directory connection to work, the Digital Video Security System host must register itself with the Yellow Pages Directory, providing a label for the Directory, and the remote computer must setup a connection using that same Yellow Pages Directory label.

4.2.1 Registering a Host with the Yellow Pages Directory

To register a Digital Video Security System local host computer with the Yellow Pages directory select the File | Register With Yellow Pages menu item or press the Register With Yellow Pages icon  on the toolbar. You will be presented with the following dialog:

Chapter 4: Reference



The Yellow Pages Registration Dialog has the following fields and controls.

Field/Control	Description
Label	Enter a label name that will appear in the Yellow Pages Directory to describe the location of your local host computer. Select a descriptive name that will be simple for you to remember. The label name is strictly used to identify a computer that has the Video Grabber attached, and does not have to correspond to any other network or other name associated with the computer's configuration.
Password	Enter a password for this Yellow Pages Directory entry. Keep the password in a safe location in case you need to unregister your Yellow Pages Directory entry.
Confirm Password	Enter the password a second time to confirm that the password was entered as intended.
Publish Label	Check this box if you want this Yellow Pages Directory entry to be available for others to look up in the Yellow Pages Directory listing. By default, this box is checked. Uncheck this box for an anonymous entry. A remote computer that wants to use your Yellow Pages entry will need to know the exact spelling of your Yellow Pages Directory entry.
Current IP Address	This is the current active IP address of your local computer. This is presented for informational purposes only.



A registered name in the Yellow Pages Directory will automatically get unregistered if it has not been used in six months. This maintains the integrity of the Yellow Pages Directory, and ensures that users do not intentionally lock up names that will not be used.

Chapter 4: Reference

4.2.2 Unregister a Host from the Yellow Pages Directory

To unregister a Digital Video Security System host computer from the Yellow Pages directory, select File | Unregister With Yellow Pages from the menu or press the

icon on the toolbar.

When a Digital Video Security System local host computer is unregistered from the Yellow Pages directory, the listing is deleted and its label is freed for another user.



ONCE YOU UNREGISTER THOSE REMOTE USERS WHO USED YOUR YELLOW PAGES ENTRY WILL NOT BE ABLE TO FIND YOUR LISTING AND THEREFORE WILL NOT BE ABLE TO CONNECT TO YOUR SYSTEM.

4.3 Setting Up Connections

The Digital Video Security System can be used to monitor security locally or remotely with equal facility. The only distinction to the user is determining which connection to log into.

Chapter 4: Reference

There are two types of Connections which you can set up:

- **Local Surveillance Connection** – A local connection is set up at the computer that has the Video Grabber, camera and Digital Video Security System software installed. It is referred to as the "Local Video" because the security camera is local to this computer. The Digital Video Security System will stream video from this location to the internet or directly to another computer, hence it is sometimes referred to as the **host computer**.
- **Remote Surveillance Connection** – A remote connection is set up at any computer that has the Digital Video Security System software installed without a camera. Use the remote connection to log into your host security camera, giving you the ability to remotely view the streamed video and to configure many of the Digital Video Security System features from a remote location.

4.3.1 The Connections Window

Both Local and Remote Surveillance Connections are set up in the Connections window. To access the Connections window, select either **View | Connections** or the

Security System window.

Connections icon from the main Digital Video

The Connections window appears as follows:

Chapter 4: Reference

The left group of icons on the Connections window toolbar is the standard View List options. The screenshot above displays the View List Details option. Please refer to Section 4.1.1.5 for more details.

The
access and modify Connections as follows:

group of icons on the right are used to add,

Icon	Description of Function
	Add New Connection: Opens the Connection dialog that is used to add both local and remote connections.

Chapter 4: Reference

	Open Connection: Opens the selected connection; provides you with access to video, Alarms and Scheduled Events for that location.
	Edit Connection: Opens the Connection dialog; used to modify existing Connection setup options.

Chapter 4: Reference

	Delete Connection: Deletes the selected Connection
--	---

Icons to the left of each Connection listed within the window denote whether it is a local or remote connection as detailed in the following table:

Connection Icons	Description
	Denotes a LOCAL connection (i.e., the camera is connected to the PC at this location)
	Denotes a REMOTE connection (i.e., will connect you to a video camera at a remote location)

Chapter 4: Reference

4.3.2 Setting Up a New Local (Host) Connection

To set up a local connection press the New Connection icon

in the Connections window, which prompts you with the dialog below. The Connection setup dialog enables you to select the desired Local connection option using the Connection Method drop-down list as shown below:

The following table describes the input required for each field of this dialog:

Connection Field	Input Value
Label	Type in any descriptive name
Type of Connection	Video Surveillance is the type of connection that is supported in this version of Digital Video Security System.
Connection Method	Select Local (TCP/IP Host) for a local connection.
Video Device	Select the desired camera from the drop-down list
Audio Device	Select the desired audio device from the drop-down list

Chapter 4: Referenc

Username	Establish a username that can be used for informational purposes when transmitting messages between a host and a remote computer. Usernames are not validated. This field is optional.
Password	Establish a password that will be required of anyone wishing to gain remote access to the connection. If set, remote users will have to enter a valid password in order to gain access to the local host. If this field is left blank, remote users will not need a password to access the local host.
OK	Saves your input values and adds the new Connection as specified.
Cancel	Closes the dialog without saving any input. Digital Video Security System prompts you to ensure that you actually intend to close the dialog without saving your changes.
Advanced	Contains advanced configuration setup options as shown in the screenshot below.

The Advanced button is used to determine the speed and port through which the video is streamed to the locations.

The Video Port is the port through which the video is streamed. The default port is 85; any other port may be selected.

The Video Profile determines the speed at which the video is streamed over the Internet. It should always be set to reflect the slower of the Internet connections of either the local host or the remote computer. It consists of the following three options representing three types of Internet connectivity:

- Slow – Modem connection to the Internet
- Medium – DSL or Cable Internet connection
- Fast – T1 or T3 Internet connection



The default connection speed is the slowest one available. Adjust this only if both the host and remote computers have faster Internet connectivity.

Chapter 4: Reference

4.3.3 Setting Up a New Remote Connection

There are two methods for establishing a remote connection to the host (local) computer:

- **Fixed IP Address** – Select this option if you know the fixed IP address of the Digital Video Security System local host computer you wish to connect to.
- **Yellow Pages Entry** – This option uses the Yellow Pages Directory service to look up the dynamic IP address of the local host. This can be used if you do not know the IP address of the local host computer you wish to connect to, or if the host is connected to an ISP service that changes its IP address on a periodic basis (many DSL and cable modem links do not maintain a guaranteed fixed IP address).

To set up a remote connection press the New Connection icon

in the Connections window, which prompts you with the dialog below. The Connection setup dialog enables you to select the desired Remote connection option using the Connection Method drop-down list as shown below:



BEWARE OF FIREWALLS! LIKE ALL APPLICATIONS, THE DIGITAL VIDEO SECURITY SYSTEM APPLICATION CANNOT PASS THROUGH A FIREWALL WITHOUT MODIFICATIONS TO THE SETTINGS OF THE FIREWALL.

Chapter 4: Reference

4.3.3.1 Remote Connection Using a Fixed IP Address

The most direct method to setup a connection is to provide the fixed IP address of the Digital Video Security System local host computer. The remote computer will establish a direct connection to the IP address when logging into the local host.



BEWARE OF MANY DSL AND CABLE MODEM CONNECTIONS THAT DO NOT HAVE A GUARANTEED FIXED ASSIGNED IP ADDRESS. EVEN IF YOUR IP ADDRESS HAS NOT CHANGED IN A WHILE, CHECK WITH YOUR ISP, DSL, OR CABLE PROVIDER TO ENSURE THAT THE IP ADDRESS IS IN FACT GUARANTEED FIXED.

To add a fixed IP address in the Connection
setup dialog, select Fixed IP Address in the Connection Method drop-down list, as shown below:

Note that the field below Connection Method now requests the fixed IP address. Simply enter the IP address of the Digital Video Security System local host computer and press OK. As noted in the table below, Login Information is optional, although recommended.

The following table describes the input required for each field of this dialog:

Connection Field	Input Value
------------------	-------------

Chapter 4: Reference

Label	Type in any descriptive name
Type of Connection	Video Surveillance is the type of connection that is supported in this version of Digital Video Security System.
Connection Method	Fixed IP Address
Fixed IP Address	Enter the Fixed IP Address of the local (host) computer.
Username	Enter a name which will be used for informational purposes only when transmitting between a remote and a local host. This field is optional.
Password	If a password has been set up at the local host, type in the required password in order to gain access to the local host.
OK	Saves your input values and adds the new Connection as specified.
Cancel	Closes the dialog without saving any input. Digital Video Security System prompts you to ensure that you actually intend to close the dialog without saving your changes.



Once a connection has been registered with the Connection Setup dialog, the Digital Video Security System enables you to easily connect to that host location by its label in the Connections window. There is no need to remember any of the connection information.

4.3.3.2 Remote Connection Using Yellow Pages Lookup

Yellow Pages Lookup is the simplest way to gain access to local host whose IP address you do not know or a host that has a dynamic IP address.



Remember that you can only use the Yellow Pages Directory to look up a host that has already been registered. Please refer to Section 4.2.1 for details on how to register a local host with the Yellow Pages.

To add a fixed IP address in the Connection setup dialog, select Fixed IP Address in the Connection Method drop-down list, as shown below:

Chapter 4: Referenc

Note that the field below Connection Method now requests the Yellow Pages Entry. If you know the Yellow Pages Directory entry of the local host you want to connect to, or if the local host registered anonymously (i.e. the local host unchecked the Publish option when registering), enter the Yellow Pages entry for the Digital Video Security System local host computer. Alternatively, you can browse the Yellow Pages Directory as outlined in Section 4.3.3.2.1. The following table describes the input required for each field of this dialog:

C nnection Field	Input Value
Label	Type in any descriptive name
Type of Connection	Video Surveillance is the type of connection that is supported in this version of Digital Video Security System.
Connection Method	Yellow Pages Entry
Yellow Pages Entry	Enter the Yellow Pages Directory entry for the local (host) computer you wish to access. Alternatively, press the button to browse the Yellow Pages Directory and select the desired host computer.
Username	Enter a name which will be used for informational purposes only when transmitting between a remote and a local host. This field is optional.

Chapter 4: Referenc

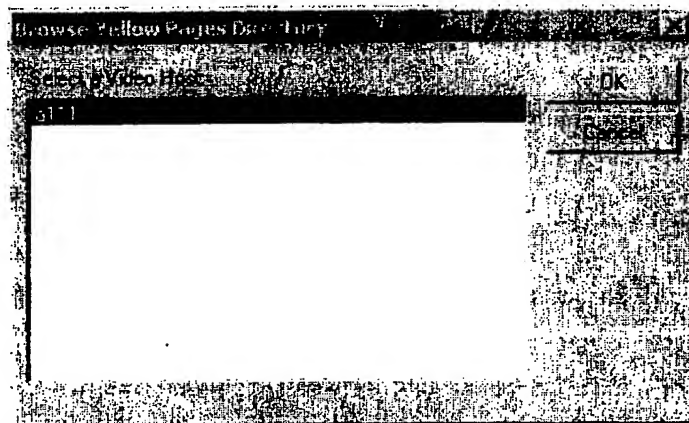
Password	If a password has been set up at the local host, type in the required password in order to gain access to the local host.
OK	Saves your input values and adds the new Connection as specified.
Cancel	Closes the dialog without saving any input. Digital Video Security System prompts you to ensure that you actually intend to close the dialog without saving your changes.

4.3.3.2.1 Browsing the Yellow Pages Directory



The Yellow Pages Directory will only list the entries of host computers that have opted to publish their listing when they registered.

To browse the Yellow Pages Directory, click the Browse button on the Connection dialog. You will be presented with the following dialog:



Chapter 4: Reference

Select the Yellow Pages entry for the Digital Video Security System host computer you are looking to connect to and press OK. The Connection dialog will be updated with the new Yellow Pages Directory.



Once you select the Yellow Pages entry from the Browse Yellow Pages Directory dialog, the current IP address of the host will be updated in your connection. You can see the IP address by selecting Fixed IP Address from the Connection Method in the Setup Connection dialog.

4.4 The Local Surveillance Window

From the Connections window (refer to Section 4.3.1), you can access the Video Surveillance Window, by simply selecting the descriptive label and pressing the

Open icon, or by double-clicking the descriptive label. The following window will open:

You can view live video in the right-hand side of the window. Alarms and Scheduled Events that have been set up for the location will be listed in the left portion of the window.

Chapter 4: Referenc

The following table describes each of the icons and their respective functions:

Surveillance Window Icon	Descripti n
	Select a List View icon to control the level of detail that is displayed in the left-hand side of the window for each Alarm or Event associated with a Connection. For details on the List View icons, refer to Section 4.1.1.5.
	The Alarm icon will open the Add New Alarm dialog, as detailed in Section 4.7.

Chapter 4: Reference

The Event icon will open the Add New Scheduled Event dialog, as detailed in Section 4.8.

Selecting this icon will delete a selected (highlighted) Alarm or Event.

Chapter 4: Referenc

	<p>The Connect icon will activate the connection, allowing remote access to the local host PC via Internet. (For more on Remote Connections, refer to Section 4.3.3.)</p>
	<p>The Disconnect icon will close the connection.</p>

Chapter 4: Reference

	Play displays the live video feed in the right-hand portion of the Local Video Surveillance window.
	Pause freezes the video playing in the right-hand portion of the Local Video Surveillance window. Press Pause again to resume live video surveillance.

Chapter 4: Reference

	Stop closes the live video feed.
	Record will save the video feed for viewing at a future time using the Video Playback feature (refer to Section 4.10).

Chapter 4: Reference

	Prompts to set the Motion Detection parameters for the currently playing video (refer to Section 4.5).
--	--

4.5 Motion Detection

The Digital Video Security System includes a powerful and flexible motion detection capability that can be calibrated to suit almost any environment. Motion detection can be defined simply as the ability to detect changes in a video stream. The Digital Video Security System essentially "memorizes" a scene and sends an alert if the scene changes.

The basic operation of motion detection is to compare each frame of a video stream with its prior frame. If there is a noticeable change then a trigger is set. If there is no change, then the frame is passed along to view.

There are many issues with motion detection that can create false alarms or not trigger motions. Some of them are:

- **Lighting** – Although humans see light as a constant, lighting in fact is continuously fluctuating. The camera sees these fluctuations and can trigger "motion" incorrectly.
- **Speed** – Depending on the circumstance, moving very quickly or very slowly can evade motion detection if it is not calibrated appropriately. If the Motion Detection is set to

Chapter 4: Reference

compare video frames every 1/10 of a second, then moving very slowly with imperceptible changes between video frames can evade detection. Likewise, if the Motion Detection is set to compare frames every 3 seconds, then moving quickly in between these checks can also evade detection.

- **Digital Fluctuations** – Digital video has natural fluctuations in how it "sees" a point in space. At any given time, the same point may be interpreted as slight variations of the same color. If improperly calibrated these slight variations can set off the alarm even when there is no actual motion.

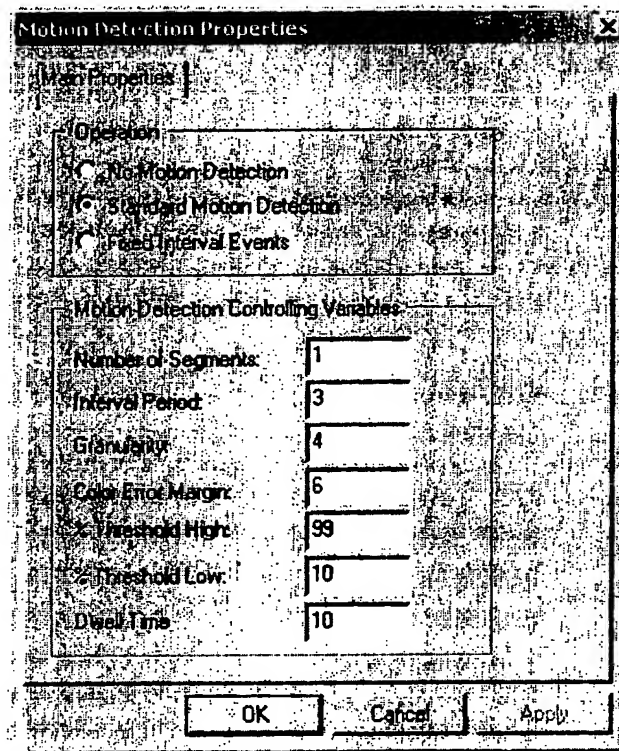
The Digital Video Security System includes a state-of-the-art motion detection capability that enables you to set the precise combination of parameters for detecting motion.

From the Local Video Surveillance window you can select the

the Motion Detection dialog:

Motion Detection icon on the toolbar to display

Chapter 4: Reference



The following table describes each of the motion detection features and their respective functions:

Motion Detection Feature	Description
Operation	By default Motion Detection is turned on whenever you start playing or recording video. You can turn motion detection off, or have it trigger at fixed intervals for testing your alarms.
Number of Segments	A video frame is broken into linear segments so that one segment at a time is checked for motion detection. If Segments is set to 1, then the whole video frame is checked for each and every frame of video. If Segments is set to 2, then half the frame is checked for motion each interval period. Motion Detection automatically rotates each segment to be checked in sequence so that the entire video frame is checked in sequence. The default value is set to 1. Change this value to conserve on CPU resources.
Interval Period	The interval is the number of one-tenth seconds between detecting for motion. The default value is .3 second.(3 intervals).
Granularity	The granularity is the number of pixels checked during frame comparisons for detecting motion. A granularity of 1 means every pixel is checked, a granularity of 2 means every second pixel is checked, etc. This value is used to increase performance and decrease CPU utilization. The default value is 4.

Chapter 4: Reference

Color Error Margin	This is the margin of error allowed, particularly for live cameras, when comparing blue with blue, red with red, etc. to determine whether there is an actual change or simply there are variations on how the colors are interpreted by the camera. Since live video can "see" the same spot each frame as slight variations of the same color, a camera might pick up the exact same spot with a slight variation in its red value in sequential frames, even though nothing has changed in the view. This is likely due to many factors, especially variations in lighting. The default value is 6.
% Threshold High	Low and high threshold determines what constitutes a change in motion. Since digital video naturally varies from frame to frame, if there is a 1% change between frames it is unlikely to be an actual change in motion. The default value for the low threshold is 5% (i.e. % changed less than 5% are not considered motion). The default value for the high threshold is 99%.
% Threshold Low	
Dwell Time	Dwell Time is the amount of time that the motion detection capability will wait until it checks to see if there is motion, from the time it first detects motion. Values are in seconds. A Dwell Time of 10 means that the motion detection will wait 10 seconds from the time it detects motion until it checks again if there is any further motion. The reason for this is to give time to react to the motion detection, rather than keep sending motion detection alarms repeatedly. The minimum value is 1 second, the maximum is 3600 seconds (1 Hour). The default value is 10 seconds.

4.6 Remote Surveillance Window

The Remote Surveillance Window is specifically designed to view video that is being streamed over the Internet from a Digital Video Security System host computer. To open the Remote Surveillance Window, select the label of the desired remote connection listed in the Connections

window (refer to Section 4.3.1) and click the Open icon - or double-click on the descriptive label. The following window will open, with the Windows Media Player logo in the right-hand side. (The Digital Video Security System uses the Windows Media Player as its video streaming technology to provide better quality and smoother Internet video streaming.)

Chapter 4: Referenc

The Remote Video Surveillance Window is comprised of four distinct areas:

The Toolbar Icons - The following table describes each of the icons and their respective functions:

Surveillance Window Icon	Description
	Select a List View icon to control the level of detail that is displayed in the left-hand side of the window for each Alarm or Event associated with a Connection. For details on the List View icons, refer to Section 4.1.1.5.

Chapter 4: Reference

The Alarm icon will open the Add New Alarm dialog, as detailed in Section 4.7.

The Event icon will open the Add New Scheduled Event dialog, as detailed in Section 4.8.

Chapter 4: Reference

	Selecting this icon will delete a selected (highlighted) Alarm or Event, as displayed in the left portion of the Remote Video Surveillance Window.
	The Connect icon will activate the connection, providing remote access to the local host PC via Internet.

Chapter 4: Referenc

The Disconnect icon will close the connection.

Windows Media Player – This is the area that is represented by the Windows Media Player, and is the actual window that displays the video. When you stretch the Remote Video Surveillance Window, the Windows Media Player stretches as well. The effect is that of zooming into a video for closer inspection. Because the video is being streamed over the Internet, there is approximately a 15 – 30 second delay before you can see the video. This speed is dependent on both the speed of your Internet connection as well as the volume of Internet traffic. Typically, the audio will descramble faster than the video, so you will hear the audio before you can see the video.

VCR-Style Controls – Under the Windows Media Player is a set of VCR-style controls that enable you to start, pause and stop video, as shown below. Use these controls when viewing live or pre-recorded video to control the video stream.



You can also control the volume using the volume control displayed below:

Feedback Bar – The Feedback Bar, located right beneath the VCR-style controls, provides feedback on the connection to a host when video is first streaming, as well as the quality of the Internet connection during streaming. When Digital Video Security System attempts to stream video from a remote host, the Feedback bar provides useful information, such as "Connecting...", "Buffering...", etc. Once a connection is established and video is being streamed from a host, the

Chapter 4: Reference

Left side of the Feedback bar displays an icon that represents the quality of the Internet connection between the remote computer and the host.

4.7 Scheduling Alarms

Digital Video Security System supports alarms based on motion detection.

To set up an alarm for Motion Detection, click on the

New Alarm icon in the Video Surveillance window (Local or Remote). The Motion Detection Alarm dialog will open, as shown below:

The following table describes each field in the top portion of the dialog. The Actions section is detailed in Section 4.9.

Motion Detection Alarm Field	Description
Alarm Description	Enter any descriptive name for the alarm.

Chapter 4: Reference

Begin Checking	Select the time you would like to activate the motion detection alarm. To change the times, position your mouse in either the hours, minutes or AM/PM field, and use the up or down arrows to scroll to your desired settings. Alternatively, you can highlight any of the fields and type the desired information in manually, one field at a time.	
End Checking	Select the time the alarm can stop monitoring for motion	
Scheduled Days	Select for which days of the week you would like the alarm to be set. By default, all weekdays will be checked. You can either click on the check boxes corresponding to the days you would like the alarm to be scheduled, or you can select one of the following convenience buttons:	
		Will select days from Monday through Friday
		Will select Saturday and Sunday

Chapter 4: Reference

		Will select all seven days
		Will clear any checked days

Alarms which have been added to a location will be displayed in the left portion of the Surveillance Window.

4.8 Scheduling Events

Establishing a Scheduled Event is nearly identical to setting an Alarm. The key difference is that rather than specifying a range of time, you are specifying the precise time you would like the event to occur.

Chapter 4: Referenc

From the Video Surveillance window (either Local or Remote), select the

Add Scheduled Event icon. This will open the following dialog, identical to the Alarm dialog with the exception of having only a defined start time (vs. a start and stop time):

Any Scheduled Event which has been added to a location will be listed in the left portion of the Video Surveillance Window.

4.9 Actions

Actions, whether in response to an Alarm trigger or as a result of a Scheduled Event, can include any of the following:

- Send a notification via e-mail
- Automatically dial a phone or pager
- Video capture
- Control X-10 devices

These actions can be set up locally or via remote connection with equal facility.

Chapter 4: Reference

To set an action associated with an Alarm or Event, simply select the desired action from the drop-down list in the New Motion Detection Alarm or New Schedule Event dialog, as shown below:

The following table describes each area of the dialog associated with Actions:

Actions Field	Description
	<p>Pressing Add will open the Add Action dialog appropriate to the Action selected from the drop-down list. Once an Action has been added, it will be listed in the Actions area window.</p>

Chapter 4: Reference

	<p>To modify an Action which is listed in the window, select (highlight) the desired Action and press the Edit button.</p>
	<p>To remove an Action from the list, select (highlight) the desired Action and press the Delete button.</p>

Chapter 4: Reference

Defines the level of detail displayed for the listed Actions associated with a given Alarm or Event. From the drop-down list, select either: Large Icons, Small Icons, List or Detail.

You can add any number of actions, in any combination, for each Alarm or Event. The following sections will detail the dialog associated with each Action.



Actions will be executed in the order listed. Although each action occurs very quickly, there may be times when you would deliberately sequence the order. For example, to in response to a motion detection Alarm, you might first capture the motion on video, and then send an e-mail with the most recent video capture as an attachment.

4.9.1 Action: Sending E-mail

The Send E-mail dialog appears as follows:

The following table describes each field in this dialog:

E-Mail Action Field	Description
To:	E-mail address of the recipient

Chapter 4: Reference

Cc:	E-mail address of any recipients receiving a copy of the e-mail.
Subject:	Subject heading for the e-mail.
Message:	Text of the e-mail message.
Attach Most Recent Video	Check this box to send the most recent video capture as an attachment to this e-mail. Especially useful in conjunction with sending an e-mail following a motion detection alarm.
OK	Adds the Action as specified in the dialog.
Cancel	Closes out the dialog without adding the specified Action. You will be asked to confirm that you want to close the dialog without saving your changes.
Test	This will immediately send the e-mail specified in the dialog. Use this feature to confirm that you have accurately completed each field in the dialog and that your e-mail reaches its intended recipient(s).



YOU MUST HAVE MAPI-COMPLIANT E-MAIL FOR THIS FEATURE TO FUNCTION.

4.9.2 Action: Dial a Phone or Pager

As a result of a triggered Alarm or a Scheduled Event, Digital Video Security System uses the currently installed TAPI-based telephone facility to dial out to a phone or pager using your modem. TAPI is the built-in Windows phone/modem facility. Most Windows modems are, in fact, TAPI-compliant. If you are unsure whether you have a TAPI-compliant modem, contact the manufacturer or refer to the modem manual.



IF YOUR SYSTEM DOES NOT HAVE A TAPI-COMPLIANT MODEM THEN THE TELEPHONE ACTION WILL NOT DIAL OUT.

The following table describes all of the fields in this dialog:

Phone/Modem Action Field	Description
---------------------------------	--------------------

Chapter 4: Reference

Phone Number	Enter the number of the phone or pager you wish to reach. You can use 9, -, *, and #. Use a comma (,) to indicate a one-second delay. For example, if you are dialing from a PBX or digital switch with a requirement to dial 9 before getting an outside line you can set the phone number to 9-,<phone number> so that after the 9 is dialed to get an outside line there is a two-second delay to allow the phone switch to connect with the outside line.
Send Dial Tones	Dial Tones are generally used to send DTMF tones to a pager. Digital Video Security System will confirm that your modem supports these DTMF tones. If this field is grayed out, as in the screen shot above, your modem does not support this feature. Send one or more telephone (DTMF) tones by entering the appropriate numbers in the message field, including 9, -, *, and #. Use a comma (,) to indicate a one-second delay.
Play Audio File	Select from one of the available audio files in the Audio File drop-down list to play an audio message once the connection to the phone/pager has been completed. For this feature to function, you must have an installed modem that supports transmission of .wav files.
Wait to Connect	This will cause the application to pause for a predetermined number of seconds before it sends out the DTMF tones and/or the audio file. This feature is included to provide you with a standard workaround for those modems which do not fully implement the TAPI standard and which return a connected signal even while still dialing out. If Digital Video Security System would send out the DTMF tones at this point, they would not reach their destination. If your modem falls under this category, use the Test feature to determine the maximum number of seconds it takes to dial out and enter the value into this field.
OK	Adds the Action as specified in the dialog.
Cancel	Closes out the dialog without adding the specified Action. You will be asked to confirm that you want to close the dialog without saving your changes.
Test	This will immediately complete the call as specified in the dialog. Use this feature to confirm that you have accurately completed each field in the dialog and that your call reaches its intended recipient(s).



A modem must be selected in the Configuration dialog for this action to function properly. Please refer to Section 4.12 for details on the Configuration dialog.



If your modem does not support DTMF tones try this workaround: In the phone number field include commas to indicate a pause when dialing. For example, if your phone system requires you to dial 9 for an outside line, and then to wait for a dial tone prior to dialing, you can enter 9,,,1-416-555-1212. Use the Test feature to accurately determine the correct number of commas needed to obtain the appropriate pause.



You can combine playing tones and an audio file in the same call. For example, you might need to send tones to get through a system with an auto-attendant and use the audio file to leave a pre-recorded message.

Chapter 4: Reference

4.9.3 Action: Record Video

To capture video as a result of an Alarm or Scheduled Event, complete the Record Video dialog:

The following table describes all of the fields in this dialog:

Video Record Action Field	Description
Title:	Any descriptive title for the recording session.
Duration:	Specify the duration of the recording session. The default value is 1 minute. Duration can range up to a maximum of 23 minutes 59 seconds. Please note that digital recordings are very disk intensive and can consume over 1 Mb of disk space per minute of recorded video.
Filename:	Specify a location and filename where the video capture will be stored.
Use Default Video Record File	As an <i>alternative</i> to specifying a filename, check this box to save the recorded video in the default Video Record file. Selecting this box will gray out the option to specify a Filename.
OK	Adds the Action as specified in the dialog.
Cancel	Closes out the dialog without adding the specified Action. You will be asked to confirm that you want to close the dialog without saving your changes.
Test	This will immediately record video as specified in the dialog. Use this feature to confirm that you have accurately completed each field in the dialog and you have recorded and saved the video as desired.

4.9.4 Action: Control X-10 Devices

Digital Video Security System can interface with X-10 controls. In order to control X-10 devices with the Digital Video Security System, you need to have a CM17a X-10 controller, otherwise known by its nickname, "FireCracker". The FireCracker device is a small connector that plugs into the Serial port of your computer and transmits an RF signal to a receiver that is plugged into the wall. In order to use the X-10 capabilities of the Digital Video Security System you need to purchase and plug the FireCracker device into a free serial port of your computer. For a brief overview of the X-10 protocol, please refer to Section 5.

Chapter 4: Reference

To control X-10 devices as a result of an Alarm or Scheduled Event, complete the following Action dialog:

The table below provides a brief description for each field option in the Control X-10 Devices dialog, along with the valid input values for each field.

Dialog Entry	Description	Valid Values
House Code	In combination with the Device Code, identifies the X-10 device(s) you would like to control.	Valid House Code values are A-P
Device Code	In combination with the House Code, identifies the X-10 device(s) you would like to control.	Valid Device Codes are 1-16
Command	Determines which command to send to the X-10 device, such as turning a light on or off.	ON, OFF or DIM
Percent Dim	Determines the level of Dim	Numeric Value
OK	Adds the Action as specified in the dialog.	
Cancel	Closes out the dialog without adding the specified Action. You will be asked to confirm that you want to close the dialog without saving your changes.	
Test	This will immediately send the command to the X-10 device as specified in the dialog. Use this feature to confirm that you have accurately completed each field in the dialog and the X-10 device respond as you intended.	

Chapter 4: Reference

4.10 Video Playback

Anytime Digital Video Security System executes a recording operation, the resulting video is saved to your computer's hard drive. You can review the recorded video anytime by clicking the

Video Playback icon on the toolbar, or selecting
View | Video Playback from the menu bar. The Video Playback window will open as shown
below:

Chapter 4: Referenc

The following table describes the functionality of each of the icons in the Video Playback window.

Video Playback Icon	Description
	Changes the default directory to look for video recording files. The files displayed in the left panel will be the ones contained in the default directory selected. Changing the directory using this option will also automatically update the Video Playback tab of the Configuration dialog (refer to Section 4.12).

Chapter 4: Reference

Deletes a selected (highlighted) video recording.

Updates the video file listing in the left panel to include any new videos which have been recorded in the Local Video Surveillance window.

Chapter 4: Reference

The toolbar contains the by-now familiar View List icons (Section 4.1.1.5), as well as the

Open,

Chapter 4: Referenc

Delete and

Update Listing icons.

The left panel contains a listing of all the video that Digital Video Security System has saved to your hard drive. To playback videos which are stored in a different directory than the one

Chapter 4: Reference

displayed, click on the **Open icon and select the** desired directory. You can modify the level of detail displayed by selecting any of the View List icons on the toolbar.

The right-hand panel of the Feedback window, containing an embedded Windows Media Player, is where the recorded video can be viewed. The Windows Media Player logo is displayed until a video is selected for viewing.

To watch a recorded video, double-click on its label in the left-hand panel. The video will begin playing in the right-hand panel. When you stretch the Video Playback window, the Windows Media Player stretches as well. The effect is that of zooming into a video for closer inspection.

Use the VCR-style controls to start, pause and stop the video, as shown below.



Use the audio controls to modify the volume:

Use the locator bar to zoom forward or backward within a video.



Chapter 4: Reference

The status bar, directly below the VCR-style controls, will display useful information about the video being played, as shown below. The numbers on the right indicate the elapsed time/the total length of the video.

Any video which has been saved by Digital Video Security System is a digital video recording and can easily be sent as an e-mail attachment or saved to disk.



Digital Video Security System supports the *.wmf file format for digital video recordings. This file format boasts an excellent compression rate and picture quality and streams well over the Internet.



Chapter 4: Reference

4.11F edback

The Feedback window, shown below, is a useful tool for monitoring which operations have been attempted and executed by Digital Video Security System.

As each operation is attempted by Digital Video Security System, the feedback window displays a listing on the status of those operations. Operations marked with a

green light have been successfully executed. For example, if you have turned on motion detection and Digital Video Security System has detected motion, the following listing will appear in the feedback window:

Chapter 4: Reference

A red light would indicate an error; an action was not carried out as requested. For example, if you scheduled an e-mail to be sent, but do not have MAPI-compliant e-mail, the feedback window would display an error message specifying that the e-mail was not sent and why.

Chapter 4: Reference

A yellow light indicates a warning.

4.12 Configuration

To access the Configuration dialog, select View | Configuration from the menu bar. You will be presented with the following dialog:

The following table describes your options in each of the tabs shown above:

Configuration Tab	Description
-------------------	-------------

Chapter 4: Reference

Connections	Determine what will happen if you double-click on a selected Connection. From the drop-down menu, select Open Connection if you would like the Connection to be opened on a double-click. Select Configure Connection if you would prefer to access the Edit Connection dialog on a double-click.
Video Playback	Use the browse button to select the default Video Playback directory. The selected directory will be used to store all video recordings, including those that result from an alarm, from a scheduled event and from selecting the record button in the Video Surveillance window.
Feedback	Determine which items of feedback to display in the Feedback window. Select to include error messages (red), warnings (yellow) and/or completed actions (green) by checking or unchecking the desired boxes.
Phone/Modem	Determine which modem to use to dial out in the Dial a Phone or Pager Action of an Alarm or Scheduled Event. Select the desired modem from the drop-down list. A modem MUST be selected in order for the Actions to execute properly.
Video Record	Determine the default duration in minutes for a video recording resulting from an Alarm or Scheduled Event. Use the up and down arrow buttons to scroll to the desired number of minutes. This will not affect recording events which are triggered by pressing the record button in the Video Surveillance window.
Yellow Pages	Determine whether Digital Video Security System should automatically register with the Yellow Pages directory upon opening.

Chapter 5: Error! Reference source not found.

5 X-10 Background Information

5.1 An Overview of the X-10 Protocol

X-10 is one of several standards for digital signals to control devices over standard 120V/240V power lines - otherwise known as a Powerline Carrier Technology ("PCT"). An X-10 device that is plugged into an outlet can control an X-10 device, such as an X-10 lamp, that is plugged into a different outlet on the same power line. The advantage of PCT is its ubiquity - all homes and offices are already wired with standard electrical power so that any device that is plugged into a wall outlet can be digitally controlled from anywhere in the facility. X-10 in particular is an open and published standard and is the leading version of PCT in North America, with vast support for X-10 devices by numerous vendors, suppliers, installers and retail outlets.

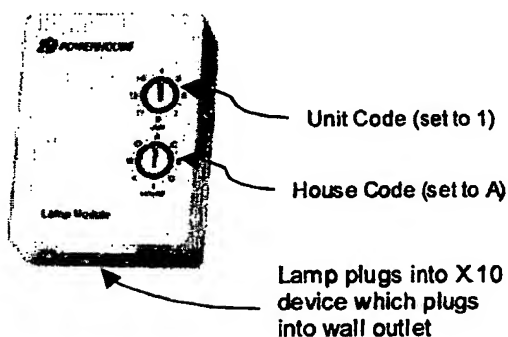


This section is for information purposes to help users who want to understand X-10. It is not necessary to understand the X-10 communication in order to use the Digital Video Security System, which hides these details from the user.

5.2 Using X-10 With Digital Video Security System

The Digital Video Security System integrates with the CM17a, or "FireCracker" X-10 controller that plugs into the serial port of your computer. The FireCracker is a popular X-10 computer interface controller because it combines a standalone remote controller with a wireless computer interface. The firecracker device itself is a small (1" x 1" x .5") device that plugs into the 9-pin COM port. It sends an RF signal to a receiver/X-10 controller that plugs into a wall outlet. This enables the computer to be located anywhere within range of the X-10 FireCracker receiver. A 4-piece FireCracker starter kit retails for approximately \$US 50.00.

Each device that implements X-10 is assigned a house code and a unit code, which is known as the device's address. The house codes are alphabetic and ranges from A to P. The unit codes are numeric and range from 1 to 16. A lamp that is plugged into the wall through an X-10 outlet will therefore carry an address such as A1 or L15. These codes are typically set manually as shown below:



As originally implemented, the house code was expected to be unique per house. It is now common practice to use multiple (or all) house codes within a single location. When X-10 is used, a command is sent to a device with a specific house and unit code, such as "turn on the device (e.g. lamp) at address A1". Any device that matches the intended address is expected to respond appropriately to the command. Commands include such things as Off, On, Dim, and Bright. It is possible to set the same address on multiple devices if you want them to all respond together.

Chapter 5: Error! Reference source not found.

5.3 Transmitting X-10 C mmands

X-10 works by encoding information on the house wiring using the 60 cycle house current as a carrier. The X-10 information is "written" at the zero crossing points (the moments in time when the 120 volt AC signal crosses 0 volts). A controller sends X-10 commands, while various receivers look for information addressed to them and take an appropriate action.

X-10 commands are simple binary values that represent the device address combined with the desired action. A multi-bit binary value (e.g. b11110) is sent to start a sequence, followed by a multi-bit device address followed by the multi-bit action command (e.g. turn on, turn off).

Typically X-10 commands are sent by standalone "controllers." A controller is simply an X-10 device that can send X-10 commands. The device below is a handheld wireless X-10 remote (NOT an X-10 controller) that sends RF commands to an X-10 controller that is plugged into the wall. The remote control is used to send an RF signal (e.g. "turn on the device at address A1") that is translated by the controller into an X-10 command and sent through the house wiring system.

6 Data Maintained by Digital Video Security System

Digital Video Security System creates and updates a number of files each time the application is accessed. All files are maintained in the directory of the application, and are transparent to the user.

To maintain persistence – that is, to ensure that Digital Video Security System maintains your settings between sessions, three files are created or updated each time you exit the program:

- The Configuration file – maintains the settings contained in the View | Configuration dialog (refer to Section 4.12). The name of this file is DVS System Configuration.dvs.
- Layout information - each time you restart the application, the layout will be just as you last left it. The name of this file is DVS System Layout.dvs.
- Connection information – maintains the values input for each connection established. The name of this file is DVS System Connections.dvs.

In addition, each time Digital Video Security System is started, a Feedback file is created and maintained until the program is exited. (Refer to Section 4.11 for further details on the Feedback window). The name of this file is DVS System Feedback.dvs.

7 Glossary

Broadband – Broadband is a general term used to refer to a fast Internet connection.

DSL – DSL is a type of Internet connection that runs over standard telephone wire.

DTMF tones – DTMF Tones are the sounds that a phone makes when you press one of the keys.

Dynamic IP – Dynamic IP is an IP Address (refer to Glossary definition of IP Address) that changes periodically.

IP Address – An IP Address is the unique address of your computer on the Internet.

ISP – ISP is the Internet Service Provider that connects your system to the Internet.

Local TCP/IP Host – A Local TCP/IP Host is the computer that is hosting a camera which is streaming video onto the Internet.

MAPI-compliant – MAPI is a standard for sending e-mail. Most E-mail systems follow the MAPI standard, such as Exchange and Outlook.

TAPI – TAPI is a standard for dialing phone numbers using a modem. A TAPI-compliant modem is one that understands TAPI signals from the computer.

.wav file – A .wav (Wave) file is a type of file that contains just recorded audio (as opposed to video).

DEMANDES OU BREVETS VOLUMINEUX

**LA PRÉSENTE PARTIE DE CETTE DEMANDE OU CE BREVETS
COMPREND PLUS D'UN TOME.**

CECI EST LE TOME 1 DE 2

NOTE: Pour les tomes additionels, veuillez contacter le Bureau Canadien des Brevets.

JUMBO APPLICATIONS / PATENTS

**THIS SECTION OF THE APPLICATION / PATENT CONTAINS MORE
THAN ONE VOLUME.**

THIS IS VOLUME 1 OF 2

NOTE: For additional volumes please contact the Canadian Patent Office.

SCHEDULE A – SOURCE CODE

```

class CYellowPages
{
};
#if
!defined(AFX_ADVCONNECTIONDLG_H__E027352A_4017_4F52_A728_02241D1FFEDE__
INCLUDED_)
#define
AFX_ADVCONNECTIONDLG_H__E027352A_4017_4F52_A728_02241D1FFEDE__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AdvConnectionDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CAdvConnectionDlg dialog

class CAdvConnectionDlg : public CDialog
{
// Construction
public:
    CAdvConnectionDlg(CWnd* pParent = NULL);    // standard
    constructor
        int m_VideoProfile;

// Dialog Data
    //{AFX_DATA(CAdvConnectionDlg)
    enum { IDD = IDD_ADVCONNECTIONDLG };
    CComboBox    m_VideoProfileCtrl;
    int          m_VideoPort;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CAdvConnectionDlg)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
    support
        //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CAdvConnectionDlg)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_ADVCONNECTIONDLG_H_E027352A_4017_4F52_A728_02241D1FFEDE__
INCLUDED_)
#if
!defined(AFX_AMTAPI_H_3428E37D_6811_4EFF_8999_A6BF26D39B8D__INCLUDED_)
#define AFX_AMTAPI_H_3428E37D_6811_4EFF_8999_A6BF26D39B8D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
// regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

////////////////////////////////////
/////
// CamTapi wrapper class

class CamTapi : public CWnd
{
protected:
    DECLARE_DYNCREATE(CamTapi)
public:
    CLSID const& GetClsid()
    {
        static CLSID const clsid
            = { 0xf270636d, 0xf062, 0x11d4, { 0x95, 0x52, 0x0,
0x40, 0x5, 0x5d, 0x80, 0x62 } };
        return clsid;
    }
    virtual BOOL Create(LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect,
        CWnd* pParentWnd, UINT nID,
        CCreateContext* pContext = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
        pParentWnd, nID); }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect, CWnd* pParentWnd, UINT nID,
        CFile* pPersist = NULL, BOOL bStorage = FALSE,
        BSTR bstrLicKey = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
        pParentWnd, nID,
        pPersist, bStorage, bstrLicKey); }

// Attributes
public:

// Operations

```

```

public:
    long GetNumberOfLines();
    long GetDevCaps();
    CString GetLineName(long Lin Id);
    void TapiReset();
    CString GetLineName();
    void SetLineName(LPCTSTR lpszNewValue);
    void About();
    void HangUp();
    void MakeCall(LPCTSTR dialNumber);
    CString GetCallState();
    void Answer();
    void ShowLocationDialog(long hWnd, LPCTSTR Number);
    CString TranslateNumber(BSTR* Number);
    long GetMediaMode();
    void SetMediaMode(long nNewValue);
    long GetCallPrivilege();
    void SetCallPrivilege(long nNewValue);
    long GetCommHandle();
    long GetLineWaveInID();
    long GetLineWaveOutID();
    void GenerateDigits(LPCTSTR Digits, long msDuration);
    void Dial(LPCTSTR dialNumber);
    CString ShowModemDialog(long hWnd, LPCTSTR DisplaySettings);
    CString GetCurrentSettings();
    void SetCurrentSettings(LPCTSTR lpszNewValue);
    BOOL GetLineOpen();
    void SetLineOpen(BOOL bNewValue);
    long GetTapiNegotiatedVersion();
    void ShowLineDialog(long hWnd);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifdef AFX_AMTAPI_H__3428E37D_6811_4EFF_8999_A6BF26D39B8D__INCLUDED_
#if
#ifdef AFX_AMWAVE_H__16720BD1_7A13_4799_9349_59DE6832087B__INCLUDED_
#define AFX_AMWAVE_H__16720BD1_7A13_4799_9349_59DE6832087B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
// regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

////////////////////////////////////
/////
// CamWave wrapper class

class CamWave : public CWnd

```

```

{
protected:
    DECLARE_DYNCREATE(CamWave)
public:
    CLSID const& GetClsid()
    {
        static CLSID const clsid
            = { 0x1a62d86f, 0x7309, 0x47d5, { 0x94, 0x77, 0x65,
0x4e, 0x76, 0x83, 0x54, 0x16 } };
        return clsid;
    }
    virtual BOOL Create(LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect,
        CWnd* pParentWnd, UINT nID,
        CCreateContext* pContext = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID); }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect, CWnd* pParentWnd, UINT nID,
        CFile* pPersist = NULL, BOOL bStorage = FALSE,
        BSTR bstrLicKey = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID,
        pPersist, bStorage, bstrLicKey); }

// Attributes
public:

// Operations
public:
    CString GetPlayFilename();
    void SetPlayFilename(LPCTSTR lpszNewValue);
    short GetPlayVolume();
    void SetPlayVolume(short nNewValue);
    CString GetRecordFilename();
    void SetRecordFilename(LPCTSTR lpszNewValue);
    short GetRecordLevel();
    void SetRecordLevel(short nNewValue);
    short GetSilenceLevel();
    void SetSilenceLevel(short nNewValue);
    short GetSilenceTimer();
    void SetSilenceTimer(short nNewValue);
    void Play(long WaveOutID);
    void Record(long WaveInID);
    void StopPlay();
    void StopRecord();
    long GetRecordFormat();
    void SetRecordFormat(long nNewValue);
    short GetWaveInNumDevs();
    short GetWaveOutNumDevs();
    CString WaveInGetName(short DeviceID);
    CString WaveOutGetName(short DeviceID);
    void About();
    long WaveInGetCaps(short DeviceID);
    long WaveOutGetCaps(short DeviceID);

```

```

    short GetPlayBufferLength();
    void SetPlayBufferLength(short nNewValue);
    short GetRecordBufferLength();
    void SetRecordBufferLength(short nNewValue);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_AMWAVE_H__16720BD1_7A13_4799_9349_59DE6832087B__INCLUDED_
#define AFX_APPSECURITYDLG_H__B2634B57_33E7_45CC_B74B_A6CF8F1A2C85__IN
    CLUDED_
#define AFX_APPSECURITYDLG_H__B2634B57_33E7_45CC_B74B_A6CF8F1A2C85__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AppSecurityDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CAppSecurityDlg dialog

class CAppSecurityDlg : public CDialog
{
// Construction
public:
    CAppSecurityDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(CAppSecurityDlg)
    enum { IDD = IDD_APPSECURITYDLG };
    CString        m_Password;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAppSecurityDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CAppSecurityDlg)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG

```

```

        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_APPSECURITYDLG_H__B2634B57_33E7_45CC_B74B_A6CF8F1A2C85__IN
    CLUDED_
#endif
#ifndef AFX_AUDIODLG_H__A0BC40BB_9E96_42E4_8912_3424CEE68AC0__INCLUDED
    _
#define AFX_AUDIODLG_H__A0BC40BB_9E96_42E4_8912_3424CEE68AC0__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AudioDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CAudioDlg dialog

class CAudioDlg : public CDialog
{
// Construction
public:
    CAudioDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{AFX_DATA(CAudioDlg)
    enum { IDD = IDD_AUDIO };
    CButton      m_Record;
    CString      m_AudioFile;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CAudioDlg)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
    support
        //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CAudioDlg)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    afx_msg void OnBrowse();
    afx_msg void OnTest();
    afx_msg void OnRecord();

```

```

        ///}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_AUDIODLG_H__A0BC40BB_9E96_42E4_8912_3424CEE68AC0__INCLUDED_
)
// ChildFrm.h : interface of the CChildFrame class
//
////////////////////////////////////
////////

#if
#ifndef AFX_CHILDFRM_H__E9482849_240E_4493_9C17_78456B7CE1A2__INCLUDED_
)
#define AFX_CHILDFRM_H__E9482849_240E_4493_9C17_78456B7CE1A2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
    // NOTE - the ClassWizard will add and remove member
    functions here.

```

```

// DO NOT EDIT what you see in these blocks of generated
code!
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_CHILDfrm_H__E9482849_240E_4493_9C17_78456B7CE1A2__INCLUDED_
#define AFX_CONNECTIONDLG_H__8CC7758F_4AFB_431C_B6CD_FC6A200B6A2A__INC
LUDED_
#define AFX_CONNECTIONDLG_H__8CC7758F_4AFB_431C_B6CD_FC6A200B6A2A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ConnectionDlg.h : header file
//

////////////////////////////////////
////////////////////////////////////
// CConnectionDlg dialog

class CConnectionDlg : public CDialog
{
// Construction
public:
    CConnectionDlg(CWnd* pParent = NULL);    // standard constructor

    CConnectionInfo* m_ConnectionInfo;
    bool m_bNewConnection;

    bool UpdateConnectionInformationControls();
    BOOL UpdateData( BOOL bSaveAndValidate = TRUE );

// Dialog Data
    {{{AFX_DATA(CConnectionDlg)
    enum { IDD = IDD_CONNECTION };
    CButton    m_MotionDetection;
    CButton    m_Advanced;
    CStatic    m_KeyDataLabelCtrl;
    CStatic    m_AudioDevicesLabel;
    CComboBox  m_VideoDevicesCtrl;
    CComboBox  m_AudioDevicesCtrl;
    CEdit m_LabelCtrl;
    CIPAddressCtrl m_IPAddress;
    int        m_ConnectionMethod;
    int        m_ConnectionType;
    }}}

```

```

        CString      m_DialUpNumber;
        CString      m_Password;
        CString      m_Username;
        CString      m_YellowPagesEntry;
        CString      m_Label;
        CString      m_Password2;
    //}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CConnectionDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CConnectionDlg)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    virtual void OnCancel();
    afx_msg void OnAdvanced();
    afx_msg void OnBrowse();
    afx_msg void OnMotiondetection();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_CONNECTIONDLG_H__8CC7758F_4AFB_431C_B6CD_FC6A200B6A2A__INC
LUDED_
#endif
#include
#define AFX_CONNECTIONSFRAME_H__7FA16ABB_61BD_4B9A_BBB4_6C7AFC135208__
INCLUDED_
#define AFX_CONNECTIONSFRAME_H__7FA16ABB_61BD_4B9A_BBB4_6C7AFC135208__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ConnectionsFrame.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CConnectionsFrame frame

class CConnectionsFrame : public CMDIChildWnd

```

```

{
    DECLARE_DYNCREATE(CConnectionsFrame)
protected:
    CConnectionsFrame();           // protected constructor used by
dynamic creation

// Attributes
public:
    CToolBar    m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CConnectionsFrame)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CConnectionsFrame();

    // Generated message map functions
    //{AFX_MSG(CConnectionsFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    afx_msg void OnDestroy();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_CONNECTIONSFRAME_H__7FA16ABB_61BD_4B9A_BBB4_6C7AFC135208__
INCLUDED_
#endif
#ifndef AFX_CONNECTIONSLISTVIEW_H__C6D196FB_0E1C_4B58_A148_A86C29E18A9
D__INCLUDED_
#define AFX_CONNECTIONSLISTVIEW_H__C6D196FB_0E1C_4B58_A148_A86C29E18A9D__INCLUD
ED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ConnectionsListView.h : header file
//

```

```

/////////////////////////////////////////////////////////////////
/////
// CConnectionsListView view

class CConnectionsListView : public CListView
{
protected:
    CConnectionsListView();          // protected constructor used
    by dynamic creation
    DECLARE_DYNCREATE(CConnectionsListView)

// Attributes
public:
    CImageList m_LargeImageList;
    CImageList m_SmallImageList;
    CImageList m_StateImageList;

// Operations
public:
    int          GetSelectedConnectionItem(bool bFeedback = true);
    bool          GetSelectedConnectionLabel(CString& strLabel);
    DWORD          GetViewType();
    BOOL          SetColumnWidth(int Column, int Width);
    BOOL          SetViewType(DWORD dwViewType);
    bool          UpdateListCtrl();

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CConnectionsListView)
    public:
        virtual void OnInitialUpdate();
    protected:
        virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    //{AFX_VIRTUAL

// Implementation
protected:
    virtual ~CConnectionsListView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(CConnectionsListView)
    afx_msg void OnViewLargeIcons();
    afx_msg void OnViewSmallIcons();
    afx_msg void OnViewDetails();
    afx_msg void OnViewList();
    afx_msg void OnAddConnection();
    afx_msg void OnDeleteConnection();
    afx_msg void OnUpdateDeleteConnection(CCmdUI* pCmdUI);

```

```

afx_msg void OnDbClick(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnOpenConnection();
afx_msg void OnUpdateOpenConnection(CCmdUI* pCmdUI);
afx_msg void OnConnect();
afx_msg void OnUpdateConnect(CCmdUI* pCmdUI);
afx_msg void OnDisconnect();
afx_msg void OnUpdateDisconnect(CCmdUI* pCmdUI);
afx_msg void OnEditConnection();
afx_msg void OnUpdateEditConnection(CCmdUI* pCmdUI);
afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_CONNECTIONSLISTVIEW_H__C6D196FB_0E1C_4B58_A148_A86C29E18A9
D__INCLUDED_)
#if
!defined(AFX_CONNECTIONSVIEW_H__F91DF593_2B94_4AD4_A877_6EA428AFD54E__I
NCLUDED_)
#define
AFX_CONNECTIONSVIEW_H__F91DF593_2B94_4AD4_A877_6EA428AFD54E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ConnectionsView.h : header file
//

////////////////////////////////////
////////
// CConnectionsView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CConnectionsView : public CFormView
{
protected:
    CConnectionsView();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CConnectionsView)

// Form Data
public:
    //{AFX_DATA(CConnectionsView)
    enum { IDD = IDD_CONNECTIONS };
    CListCtrl m_ConnectionsList;
    //}}AFX_DATA

```

```

// Attributes
public:

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CConnectionsView)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CConnectionsView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
    //{{AFX_MSG(CConnectionsView)
    // NOTE - the ClassWizard will add and remove member
functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_CONNECTIONSVIEW_H__F91DF593_2B94_4AD4_A877_6EA428AFD54E__I
NCLUDED_

#define CM17ACOMMAND_ON        0x02
#define CM17ACOMMAND_OFF      0x03
#define CM17ACOMMAND_DIM      0x04
#define CM17ACOMMAND_BRIGHT  0x05

#import "cm17a.ocx" named_guids no_namespace

class CX10
{
protected:
    _controlcm*          m_pCM17;
    short               m_nPort;

public:
    CX10();

```

```

        bool        Initialize();
        bool        ExecuteCommand(LPSTR szHouseCode, LPSTR szDeviceCode,
short Command, short Brightness);
        bool        ExecuteCommand(int nHouseCode, int nDeviceCode, short
Command, short Brightness);
        bool        SetPort(short nPort);
        bool        Uninitialize();
};

#pragma once

class CConnectionInfo;
class CEventInfo;

enum enum_EVENTTYPE
{
    EVENTTYPE_ALARM,
    EVENTTYPE_SCHEDULEDEVENT
};

enum enum_ACTIONTYPE
{
    ACTIONTYPE_EMAIL,
    ACTIONTYPE_PHONE,
    ACTIONTYPE_RECORDVIDEO,
    ACTIONTYPE_X10,
    ACTIONTYPE_AUDIO,
};

enum enum_CONNECTIONMETHOD
{
    CONNECTION_LOCALTCPIP,
    // CONNECTION_LOCALMODEM,
    CONNECTION_IPADDRESS,
    CONNECTION_YELLOWPAGES,
    // CONNECTION_DIALUP,
};

enum enum_CONNECTIONTYPE
{
    CONTYPE_VIDEOSURVEILLANCE,
    CONTYPE_INSTANTMESSENGER,
    CONTYPE_VIDEOCONFERENCE
};

class CActionInfo : public CObject
{
public:
    int                m_ActionType;

public:
    // DECLARE_SERIAL( CActionInfo )
    CActionInfo();

    virtual bool        EditAction() = NULL;

```

```

        virtual bool      GetActionSummary(CString& strSummary) = NULL;
        virtual void      Serialize( CArchive& archive ) = NULL;
        virtual bool      Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo) = NULL;
    };

class CAudioAction : public CActionInfo
{
public:
    CString          m_AudioFile;

public:
    CAudioAction() ;
    CAudioAction(CAudioAction* CAudioAction);

    bool            EditAction();
    bool            ExecuteAction();
    bool            GetActionSummary(CString& strSummary);
    void            Serialize( CArchive& archive );
    bool            Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
};

class CEmailAction : public CActionInfo
{
public:
    CString          m_To;
    CString          m_Cc;
    CString          m_Subject;
    CString          m_Message;
    BOOL            m_AttachVideo;
    CTime            m_Duration;
    BOOL            m_KeepLeadingVideo;

public:
    CEmailAction();
    CEmailAction(CEmailAction* EMailAction);

    bool            EditAction();
    bool            ExecuteAction(CConnectionInfo* pConnectionInfo);
    bool            GetActionSummary(CString& strSummary);
    CEmailAction& operator= ( CEmailAction& EMailAction );
    void            Serialize( CArchive& archive );
    bool            Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
};

class CPhoneAction : public CActionInfo
{
public:
    CString          m_AudioFile;
    CString          m_DialTones;
    CString          m_PhoneNumber;
    DWORD m_WaitToHangUp;

public:
    CPhoneAction();

```

```

CPhoneAction(CPhoneAction * PhoneAction);

bool        EditAction();
bool        ExecuteAction();
bool        GetActionSummary(CString& strSummary);
void        Serialize( CArchive& archive );
bool        Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
};

class CVideoRecordAction : public CActionInfo
{
public:
    CTime        m_Duration;
    CString      m_Filename;
    BOOL         m_UseDefaultFilename;
    CString      m_Title;
    BOOL         m_KeepLeadingVideo;
    BOOL         m_Continuous;
    BOOL         m_OverwritePrior;

public:
    CVideoRecordAction();
    CVideoRecordAction(CVideoRecordAction * VideoRecordAction);

    bool        EditAction();
    bool        ExecuteAction(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo);
    bool        GetActionSummary(CString& strSummary);
    void        Serialize( CArchive& archive );
    bool        Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
};

enum enum_X10COMMAND
{
    X10COMMAND_ON,
    X10COMMAND_OFF,
    X10COMMAND_DIM
};

class CX10Action : public CActionInfo
{
public:
    int         m_Command;
    int         m_DeviceCode;
    int         m_HouseCode;
    int         m_PercentDim;
    short m_Port;

public:
    CX10Action();
    CX10Action(CX10Action * X10Action);

    bool        EditAction();
    bool        ExecuteAction();
    bool        GetActionSummary(CString& strSummary);

```

```

        void        Serialize( CArchive& archive );
        bool        Trigger(CConn ctionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
    };

```

```

class CActionsArray : public CObArray
{
public:
    CActionsArray();
    ~CActionsArray();

    bool        AddAction(int nActionType);
    bool        AddAudioAction();
    bool        AddEMailAction();
    bool        AddPhoneAction();
    bool        AddVideoRecordAction();
    bool        AddX10Action();
    bool        DeleteAction(int nActionIndex);
    void        DeleteAll();
    bool        EditAction(int nActionIndex);
    void        Serialize( CArchive& archive );
    bool        Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo);
};

```

```

class CEventInfo : public CObject
{
public:
    int          m_EventType;
    CString      m_Label;
    CTime        m_StartTime;
    CTime        m_EndTime;
    BOOL         m_Sunday;
    BOOL         m_Monday;
    BOOL         m_Tuesday;
    BOOL         m_Wednesday;
    BOOL         m_Thursday;
    BOOL         m_Friday;
    BOOL         m_Saturday;

    CActionsArray m_Actions;

public:
    DECLARE_SERIAL( CEventInfo )
    CEventInfo();
    CEventInfo& operator= ( CEventInfo& EventInfo);

    bool        GetEventDetailString(CString& t_str);
    bool        GetEventTypeString(CString& strEventType);
    void        Serialize( CArchive& archive );
    bool        TriggerAlarm(CConnectionInfo* pConnectionInfo);
    bool        TriggerScheduledEvent(CTime t_CurrentTime,
CConnectionInfo* pConnectionInfo);
};

```

```

class CEventsArray : public CArray<CEventInfo, CEventInfo&>
{
public:
    CEventsArray();

    bool        DeleteEvent(CString strEventLabel);
    bool        EditEvent(CString strConnectionLabel, CString
strEventLabel);
    int          GetIndexFromLabel(CString strEventLabel);
    bool        IsLabelValid(CString strLabel);
    void        Serialize( CArchive& archive );
    bool        TriggerScheduledEvent(CTime t_CurrentTime,
CConnectionInfo* pConnectionInfo);
};

//*****
//
// Description:   Simplifies working with DirectPlay
//
//*****
enum enum_SIMPLEDIRECTPLAYSTATE
{
    SIMPLEDIRECTPLAYSTATE_INACTIVE,
    SIMPLEDIRECTPLAYSTATE_ACTIVE,
};

enum enum_SDPSERVICETYPE
{
    SDPSERVICETYPE_TCPIP,
    SDPSERVICETYPE_MODEM,
};

class CSimpleDirectPlay
{
//
// DirectPlay variables and functions
//
protected:
    // DirectPlay objects
    IDirectPlay8Peer*      m_pDP;      // DirectPlay peer object
    IDirectPlay8Address*   m_pDeviceAddressLocal ;
    IDirectPlay8Address*   m_pDeviceAddressTarget ;

```

```

// Internal DirectPlay wrapper functions
bool      ConnectToSession();
bool      GetServiceProviderGuid(GUID& guid);
bool      GetURLAString(CString& strURLA);
bool      HostSession();
bool      InitializeDirectPlayPeer();
bool      SetupLocalDPAddress();
bool      SetupTargetDPAddress();

/////////////////////////////////////////////////////////////////
/////
//
// SimpleDirectPlay public variables and functions
//
/////////////////////////////////////////////////////////////////
/////
public:
    CSimpleDirectPlay();

    // Public variables
    CString      m_strSessionName;
    CString      m_strIPAddress;
    CString      m_strPassword;
    UINT         m_State;
    UINT         m_ServiceType;

    // Public functions
    bool         ConnectHostSession();
    bool         ConnectRemoteSession();
    bool         DisconnectHostSession();
    bool         DisconnectRemoteSession();
    bool         SendMessage(CString strMessage);
    bool         SendMessage(CConnectionMsg& ConnectionMsg);
    bool         IsInactive();

};

class CConnectionInfo : public CObject
{
public:
    ///////////////////////////////////////////////////////////////////
    /////
    //
    // Connection variables and functions
    //
    //
    // Note: For every variable added, make sure that it is included in
    //       1) GetCopy
    //       2) SetCopy
    //       2) Initialize
    //       2) Serialize
    //
    ///////////////////////////////////////////////////////////////////
    /////

    CString      m_Label;

```

```

CString      m_IPAddress;
CString      m_YellowPagesEntry;
CString      m_DialUpNumber;
UINT         m_ConnectionMethod;
UINT         m_ConnectionType;
CString      m_Username;
CString      m_Password;

// List of all alarms and scheduled events
CEventsArray m_Events;

// Controls all video functionality
CSimpleVideo m_SimpleVideo;

// Controls all data communication functionality via DirectPlay
CSimpleDirectPlay m_SimpleDirectPlay;

public:
    DECLARE_SERIAL( CConnectionInfo )
    CConnectionInfo();
    ~CConnectionInfo();
    CConnectionInfo(CConnectionInfo& ConnectionInfo);
    CConnectionInfo& operator = (CConnectionInfo& ConnectionInfo);

    bool      AddEvent(int nEventType);
    bool      Connect(HWND hWnd);
    bool      Disconnect();
    bool      GetConnectionKeyData(CString& KeyData );
    bool      GetConnectionMethodString(CString& ConnectionMethod);
    bool      GetConnectionTypeString(CString& ConnectionType);
    void      GetCopy(CConnectionInfo& ConnectionInfo);
    bool      GetIPAddress(CString& strIPAddress);
    void      Initialize();
    bool      IsConnectionEstablished();
    bool      IsEventLabelValid(CString strEventLabel);
    bool      ReceiveEvents(CConnectionMsg* pConnectionMsg);
    bool      SendEvents();
    void      Serialize( CArchive& archive );
    void      SetCopy(CConnectionInfo& ConnectionInfo);
    bool      TriggerAlarm();
    bool      TriggerScheduledEvent(CTime t_CurrentTime);

};

class CConnectionsArray : public CArray<CConnectionInfo,
CConnectionInfo&>
{
public:
    CConnectionsArray() {};

    bool      AddConnection();
    bool      AddEvent(CString strLabel, int nEventType);
    bool      EditEvent(CString strConnectionLabel, CString
strEventLabel);
    bool      Connect(CString strLabel, HWND hWnd);
    void      Debug();
    bool      DeleteConnection(CString strLabel);

```

```

        bool        DeleteEvent(CString strConnectionLabel, CString
strEventLabel);
        bool        Disconnect(CString strLabel);
        bool        IsConnectionEstablished(CString strLabel);
        bool        IsEventLabelValid(CString strConnectionLabel, CString
strEventLabel);
        bool        IsLabelValid(CString strLabel);
        bool        EditConnection(CString strLabel);
        bool        GetAudioDevice(CString strLabel, CString&
strAudioDevice);
        bool        GetConnectionMethod(CString strLabel, UINT&
ConnectionMethod);
        bool        GetConnectionType(CString strLabel, UINT&
ConnectionType);
        int         GetIndexFromLabel(CString strLabel);
        bool        GetIPAddress(CString strLabel, CString&
strIPAddress);
        bool        GetPort(CString strLabel, DWORD& dwPort);
        bool        GetSerializeFileName(CString& strFileName);
        bool        GetSimpleVideo(CString strLabel, UINT& State);
        bool        GetVideoDevice(CString strLabel, CString&
strVideoDevice);
        bool        IsValidIndex(int nIndex);
        bool        LoadSettings();
        bool        ReceiveEvents(CString strLabel, CConnectionMsg*
pConnectionMsg);
        void        SaveSettings();
        bool        SendEvents(CString strLabel);
        bool        SendMessage(CString strLabel, CString strMessage);
        bool        SendMessage(CString strLabel, CConnectionMsg&
ConnectionMsg);
        void        Serialize(CArchive& archive);
        bool        SerializeEvents(CString strLabel, CArchive& archive);
        bool        TriggerAlarm(CString strLabel);
        bool        TriggerScheduledEvent(CTime t_CurrentTime);
};

//
// COM definition for DirectPlay8 Peer interface
//
#undef INTERFACE // External COM Implementation
#define INTERFACE IDirectPlay8Peer
DECLARE_INTERFACE_(IDirectPlay8Peer, IUnknown)
{
    /*** IUnknown methods ***/
    STDMETHOD(QueryInterface)                (THIS_ REFIID riid,
LPVOID *ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)                (THIS) PURE;
    STDMETHOD_(ULONG, Release)               (THIS) PURE;
    /*** IDirectPlay8Peer methods ***/
    STDMETHOD(Initialize)                   (THIS_ PVOID const
pvUserContext, const PFNDPNMESSAGEHANDLER pfn, const DWORD dwFlags)
PURE;
    STDMETHOD(EnumServiceProviders)         (THIS_ const GUID
*const pguidServiceProvider, const GUID *const pguidApplication,
DPN_SERVICE_PROVIDER_INFO *const pSPInfoBuffer, DWORD *const
pcbEnumData, DWORD *const pcReturned, const DWORD dwFlags) PURE;

```

```

        STDMETHODCALLTYPE (CancelAsyncOperation) (THIS_ const DPNHANDLE
hAsyncHandle, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (Connect) (THIS_ const
DPN_APPLICATION_DESC *const pdnAppDesc, IDirectPlay8Address *const
pHostAddr, IDirectPlay8Address *const pDeviceInfo, const
DPN_SECURITY_DESC *const pdnSecurity, const DPN_SECURITY_CREDENTIALS
*const pdnCredentials, const void *const pvUserConnectData, const DWORD
dwUserConnectDataSize, void *const pvPlayerContext, void *const
pvAsyncContext, DPNHANDLE *const phAsyncHandle, const DWORD dwFlags)
PURE;

        STDMETHODCALLTYPE (SendTo) (THIS_ const DPNID
dpnid, const DPN_BUFFER_DESC *const prgBufferDesc, const DWORD
cBufferDesc, const DWORD dwTimeOut, void *const pvAsyncContext, DPNHANDLE
*const phAsyncHandle, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (GetSendQueueInfo) (THIS_ const DPNID
dpnid, DWORD *const pdwNumMsgs, DWORD *const pdwNumBytes, const DWORD
dwFlags) PURE;

        STDMETHODCALLTYPE (Host) (THIS_ const
DPN_APPLICATION_DESC *const pdnAppDesc, IDirectPlay8Address **const
prgpDeviceInfo, const DWORD cDeviceInfo, const DPN_SECURITY_DESC *const
pdnSecurity, const DPN_SECURITY_CREDENTIALS *const pdnCredentials, void
*const pvPlayerContext, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (GetApplicationDesc) (THIS_ DPN_APPLICATION_DESC
*const pAppDescBuffer, DWORD *const pcbDataSize, const DWORD dwFlags)
PURE;

        STDMETHODCALLTYPE (SetApplicationDesc) (THIS_ const
DPN_APPLICATION_DESC *const pad, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (CreateGroup) (THIS_ const
DPN_GROUP_INFO *const pdpnGroupInfo, void *const pvGroupContext, void
*const pvAsyncContext, DPNHANDLE *const phAsyncHandle, const DWORD
dwFlags) PURE;

        STDMETHODCALLTYPE (DestroyGroup) (THIS_ const DPNID
idGroup, PVOID const pvAsyncContext, DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (AddPlayerToGroup) (THIS_ const DPNID
idGroup, const DPNID idClient, PVOID const pvAsyncContext, DPNHANDLE
*const phAsyncHandle, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (RemovePlayerFromGroup) (THIS_ const DPNID idGroup,
const DPNID idClient, PVOID const pvAsyncContext, DPNHANDLE *const
phAsyncHandle, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (SetGroupInfo) (THIS_ const DPNID
dpnid, DPN_GROUP_INFO *const pdpnGroupInfo, PVOID const
pvAsyncContext, DPNHANDLE *const phAsyncHandle, const DWORD dwFlags)
PURE;

        STDMETHODCALLTYPE (GetGroupInfo) (THIS_ const DPNID
dpnid, DPN_GROUP_INFO *const pdpnGroupInfo, DWORD *const pdwSize, const
DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (EnumPlayersAndGroups) (THIS_ DPNID *const
prgdpnid, DWORD *const pcdpnid, const DWORD dwFlags) PURE;

        STDMETHODCALLTYPE (EnumGroupMembers) (THIS_ const DPNID
dpnid, DPNID *const prgdpnid, DWORD *const pcdpnid, const DWORD
dwFlags) PURE;

        STDMETHODCALLTYPE (SetPeerInfo) (THIS_ const
DPN_PLAYER_INFO *const pdpnPlayerInfo, PVOID const
pvAsyncContext, DPNHANDLE *const phAsyncHandle, const DWORD dwFlags)
PURE;

```

```

        STDMETHODCALLTYPE(GetPeerInfo)                (THIS_ const DPNID
dpnid, DPN_PLAYER_INFO *const pdpnPlayerInfo, DWORD *const pdwSize, const
DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetPeerAddress)            (THIS_ const DPNID
dpnid, IDirectPlay8Address **const pAddress, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetLocalHostAddresses)      (THIS_ IDirectPlay8Address
**const prgpAddress, DWORD *const pcAddress, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(Close)                    (THIS_ const DWORD
dwFlags) PURE;
        STDMETHODCALLTYPE(EnumHosts)                (THIS_
PDPN_APPLICATION_DESC const pApplicationDesc, IDirectPlay8Address *const
pAddrHost, IDirectPlay8Address *const pDeviceInfo, PVOID const
pUserEnumData, const DWORD dwUserEnumDataSize, const DWORD
dwEnumCount, const DWORD dwRetryInterval, const DWORD dwTimeOut, PVOID
const pvUserContext, DPNHANDLE *const pAsyncHandle, const DWORD dwFlags)
PURE;
        STDMETHODCALLTYPE(DestroyPeer)              (THIS_ const DPNID
dpnidClient, const void *const pvDestroyData, const DWORD
dwDestroyDataSize, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(ReturnBuffer)              (THIS_ const DPNHANDLE
hBufferHandle, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetPlayerContext)          (THIS_ const DPNID
dpnid, PVOID *const ppvPlayerContext, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetGroupContext)           (THIS_ const DPNID
dpnid, PVOID *const ppvGroupContext, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetCaps)                  (THIS_ DPN_CAPS
*const pdpCaps, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(SetCaps)                  (THIS_ const
DPN_CAPS *const pdpCaps, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(SetSPCaps)                (THIS_ const GUID * const
pguidSP, const DPN_SP_CAPS *const pdpspCaps, const DWORD dwFlags )
PURE;
        STDMETHODCALLTYPE(GetSPCaps)                (THIS_ const GUID * const
pguidSP, DPN_SP_CAPS *const pdpspCaps, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(GetConnectionInfo)        (THIS_ const DPNID dpnid,
DPN_CONNECTION_INFO *const pdpConnectionInfo, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(RegisterLobby)            (THIS_ const DPNHANDLE
dpnHandle, struct IDirectPlay8LobbiedApplication *const
pIDP8LobbiedApplication, const DWORD dwFlags) PURE;
        STDMETHODCALLTYPE(TerminateSession)         (THIS_ void *const
pvTerminateData, const DWORD dwTerminateDataSize, const DWORD dwFlags)
PURE;
};

#define      IDirectPlay8Peer_QueryInterface(p,a,b)
(p)->QueryInterface(a,b)
#define      IDirectPlay8Peer_AddRef(p)
(p)->AddRef()
#define      IDirectPlay8Peer_Release(p)
(p)->Release()
#define      IDirectPlay8Peer_Initialize(p,a,b,c)
(p)->Initialize(a,b,c)
#define      IDirectPlay8Peer_EnumServiceProviders(p,a,b,c,d,e,f) (p)-
>EnumServiceProviders(a,b,c,d,e,f)
#define      IDirectPlay8Peer_EnumHosts(p,a,b,c,d,e,f,g,h,i,j,k)
(p)->EnumHosts(a,b,c,d,e,f,g,h,i,j,k)

```

```

#define      IDirectPlay8Peer_CancelAsyncOperation(p,a,b)
(p)->CancelAsyncOperation(a,b)
#define      IDirectPlay8Peer_Connect(p,a,b,c,d,e,f,g,h,i,j,k)
(p)->Connect(a,b,c,d,e,f,g,h,i,j,k)
#define      IDirectPlay8Peer_SendTo(p,a,b,c,d, ,f,g)
(p)->SendTo(a,b,c,d,e,f,g)
#define      IDirectPlay8Peer_GetSendQueueInfo(p,a,b,c,d)
(p)->GetSendQueueInfo(a,b,c,d)
#define      IDirectPlay8Peer_Host(p,a,b,c,d,e,f,g)
(p)->Host(a,b,c,d,e,f,g)
#define      IDirectPlay8Peer_GetApplicationDesc(p,a,b,c)
(p)->GetApplicationDesc(a,b,c)
#define      IDirectPlay8Peer_SetApplicationDesc(p,a,b)
(p)->SetApplicationDesc(a,b)
#define      IDirectPlay8Peer_CreateGroup(p,a,b,c,d,e)
(p)->CreateGroup(a,b,c,d,e)

#define      IDirectPlay8Peer_DestroyGroup(p,a,b,c,d)
(p)->DestroyGroup(a,b,c,d)
#define      IDirectPlay8Peer_AddPlayerToGroup(p,a,b,c,d,e)
(p)->AddPlayerToGroup(a,b,c,d,e)
#define      IDirectPlay8Peer_RemovePlayerFromGroup(p,a,b,c,d,e)
(p)->RemovePlayerFromGroup(a,b,c,d,e)
#define      IDirectPlay8Peer_SetGroupInfo(p,a,b,c,d,e)
(p)->SetGroupInfo(a,b,c,d,e)
#define      IDirectPlay8Peer_GetGroupInfo(p,a,b,c,d)
(p)->GetGroupInfo(a,b,c,d)
#define      IDirectPlay8Peer_EnumPlayersAndGroups(p,a,b,c)
(p)->EnumPlayersAndGroups(a,b,c)
#define      IDirectPlay8Peer_EnumGroupMembers(p,a,b,c,d)
(p)->EnumGroupMembers(a,b,c,d)
#define      IDirectPlay8Peer_SetPeerInfo(p,a,b,c,d)
(p)->SetPeerInfo(a,b,c,d)
#define      IDirectPlay8Peer_GetPeerInfo(p,a,b,c,d)
(p)->GetPeerInfo(a,b,c,d)
#define      IDirectPlay8Peer_GetPeerAddress(p,a,b,c)
(p)->GetPeerAddress(a,b,c)
#define      IDirectPlay8Peer_GetLocalHostAddresses(p,a,b,c)
(p)->GetLocalHostAddresses(a,b,c)
#define      IDirectPlay8Peer_Close(p,a)
(p)->Close(a)
#define      IDirectPlay8Peer_EnumHosts(p,a,b,c,d,e,f,g,h,i,j,k)
(p)->EnumHosts(a,b,c,d,e,f,g,h,i,j,k)
#define      IDirectPlay8Peer_DestroyPeer(p,a,b,c,d)
(p)->DestroyPeer(a,b,c,d)
#define      IDirectPlay8Peer_ReturnBuffer(p,a,b)
(p)->ReturnBuffer(a,b)
#define      IDirectPlay8Peer_GetPlayerContext(p,a,b,c)
(p)->GetPlayerContext(a,b,c)
#define      IDirectPlay8Peer_GetGroupContext(p,a,b,c)
(p)->GetGroupContext(a,b,c)
#define      IDirectPlay8Peer_GetCaps(p,a,b)
(p)->GetCaps(a,b)
#define      IDirectPlay8Peer_SetCaps(p,a,b)
(p)->SetCaps(a,b)
#define      IDirectPlay8Peer_SetSPCaps(p,a,b,c)
(p)->SetSPCaps(a,b,c)

```

```

#define IDirectPlay8Peer_GetSPCaps(p,a,b,c)
    (p)->GetSPCaps(a,b,c)
#define IDirectPlay8Peer_GetConnectionInfo(p,a,b,c)
    (p)->GetConnectionInfo(a,b,c)
#define IDirectPlay8Peer_RegisterLobby(p,a,b,c)
    (p)->RegisterLobby(a,b,c)
#define IDirectPlay8Peer_TerminateSession(p,a,b,c)
    (p)->TerminateSession(a,b,c)

#ifndef __ICaptureGraphBuilder2_FWD_DEFINED__
#define __ICaptureGraphBuilder2_FWD_DEFINED__
typedef interface ICaptureGraphBuilder2 ICaptureGraphBuilder2;
#endif /* __ICaptureGraphBuilder2_FWD_DEFINED__ */

#ifndef __ICaptureGraphBuilder2_INTERFACE_DEFINED__
#define __ICaptureGraphBuilder2_INTERFACE_DEFINED__

/* interface ICaptureGraphBuilder2 */
/* [unique][uuid][object] */

EXTERN_C const IID IID_ICaptureGraphBuilder2;

#if defined(__cplusplus) && !defined(CINTERFACE)

MIDL_INTERFACE("93E5A4E0-2D50-11d2-ABFA-00A0C9C6E38D")
ICaptureGraphBuilder2 : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE SetFiltergraph(
        /* [in] */ IGraphBuilder *pfg) = 0;

    virtual HRESULT STDMETHODCALLTYPE GetFiltergraph(
        /* [out] */ IGraphBuilder **ppfg) = 0;

    virtual HRESULT STDMETHODCALLTYPE SetOutputFileName(
        /* [in] */ const GUID *pType,
        /* [in] */ LPCOLESTR lpstrFile,
        /* [out] */ IBaseFilter **ppf,
        /* [out] */ IFileSinkFilter **ppSink) = 0;

    virtual /* [local] */ HRESULT STDMETHODCALLTYPE FindInterface(
        /* [in] */ const GUID *pCategory,
        /* [in] */ const GUID *pType,
        /* [in] */ IBaseFilter *pf,
        /* [in] */ REFIID riid,
        /* [out] */ void **ppint) = 0;

    virtual HRESULT STDMETHODCALLTYPE RenderStream(
        /* [in] */ const GUID *pCategory,
        /* [in] */ const GUID *pType,
        /* [in] */ IUnknown *pSource,
        /* [in] */ IBaseFilter *pfCompressor,
        /* [in] */ IBaseFilter *pfRenderer) = 0;

    virtual HRESULT STDMETHODCALLTYPE ControlStream(

```

```

        /* [in] */ const GUID *pCategory,
        /* [in] */ const GUID *pType,
        /* [in] */ IBaseFilter *pFilter,
        /* [in] */ REFERENCE_TIME *pstart,
        /* [in] */ REFERENCE_TIME *pstop,
        /* [in] */ WORD wStartCookie,
        /* [in] */ WORD wStopCookie) = 0;

virtual HRESULT STDMETHODCALLTYPE AllocCapFile(
    /* [in] */ LPCOLESTR lpstr,
    /* [in] */ DWORDLONG dwlSize) = 0;

virtual HRESULT STDMETHODCALLTYPE CopyCaptureFile(
    /* [in] */ LPOLESTR lpwstrOld,
    /* [in] */ LPOLESTR lpwstrNew,
    /* [in] */ int fAllowEscAbort,
    /* [in] */ IAMCopyCaptureFileProgress *pCallback) = 0;

virtual HRESULT STDMETHODCALLTYPE FindPin(
    /* [in] */ IUnknown *pSource,
    /* [in] */ PIN_DIRECTION pindir,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ BOOL fUnconnected,
    /* [in] */ int num,
    /* [out] */ IPin **ppPin) = 0;

};

#else          /* C style interface */

typedef struct ICaptureGraphBuilder2Vtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        ICaptureGraphBuilder2 * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        ICaptureGraphBuilder2 * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        ICaptureGraphBuilder2 * This);

    HRESULT ( STDMETHODCALLTYPE *SetFiltergraph )(
        ICaptureGraphBuilder2 * This,
        /* [in] */ IGraphBuilder *pfg);

    HRESULT ( STDMETHODCALLTYPE *GetFiltergraph )(
        ICaptureGraphBuilder2 * This,
        /* [out] */ IGraphBuilder **ppfg);

    HRESULT ( STDMETHODCALLTYPE *SetOutputFileName )(
        ICaptureGraphBuilder2 * This,
        /* [in] */ const GUID *pType,

```

```

    /* [in] */ LPCOLESTR lpstrFile,
    /* [out] */ IBaseFilter **ppf,
    /* [out] */ IFileSinkFilter **ppSink);

/* [local] */ HRESULT ( STDMETHODCALLTYPE *FindInterface )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ IBaseFilter *pf,
    /* [in] */ REFIID riid,
    /* [out] */ void **ppint);

HRESULT ( STDMETHODCALLTYPE *RenderStream )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ IUnknown *pSource,
    /* [in] */ IBaseFilter *pfCompressor,
    /* [in] */ IBaseFilter *pfRenderer);

HRESULT ( STDMETHODCALLTYPE *ControlStream )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ IBaseFilter *pFilter,
    /* [in] */ REFERENCE_TIME *pstart,
    /* [in] */ REFERENCE_TIME *pstop,
    /* [in] */ WORD wStartCookie,
    /* [in] */ WORD wStopCookie);

HRESULT ( STDMETHODCALLTYPE *AllocCapFile )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ LPCOLESTR lpstr,
    /* [in] */ DWORDLONG dwlSize);

HRESULT ( STDMETHODCALLTYPE *CopyCaptureFile )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ LPOLESTR lpwstrOld,
    /* [in] */ LPOLESTR lpwstrNew,
    /* [in] */ int fAllowEscAbort,
    /* [in] */ IAMCopyCaptureFileProgress *pCallback);

HRESULT ( STDMETHODCALLTYPE *FindPin )(
    ICaptureGraphBuilder2 * This,
    /* [in] */ IUnknown *pSource,
    /* [in] */ PIN_DIRECTION pindir,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ BOOL fUnconnected,
    /* [in] */ int num,
    /* [out] */ IPin **ppPin);

    END_INTERFACE
} ICaptureGraphBuilder2Vtbl;

interface ICaptureGraphBuilder2
{

```

```

        CONST_VTBL struct ICaptureGraphBuilder2Vtbl *lpVtbl;
};

#ifdef COBJMACROS

#define ICaptureGraphBuilder2_QueryInterface(This,riid,ppvObject) \
    (This)->lpVtbl->QueryInterface(This,riid,ppvObject)

#define ICaptureGraphBuilder2_AddRef(This) \
    (This)->lpVtbl->AddRef(This)

#define ICaptureGraphBuilder2_Release(This) \
    (This)->lpVtbl->Release(This)

#define ICaptureGraphBuilder2_SetFiltergraph(This,pfg) \
    (This)->lpVtbl->SetFiltergraph(This,pfg)

#define ICaptureGraphBuilder2_GetFiltergraph(This,ppfg) \
    (This)->lpVtbl->GetFiltergraph(This,ppfg)

#define ICaptureGraphBuilder2_SetOutputFileName(This,pType,lpstrFile,ppf,ppSink) \
    (This)->lpVtbl->SetOutputFileName(This,pType,lpstrFile,ppf,ppSink)

#define ICaptureGraphBuilder2_FindInterface(This,pCategory,pType,pf,riid,ppint) \
    (This)->lpVtbl->FindInterface(This,pCategory,pType,pf,riid,ppint)

#define ICaptureGraphBuilder2_RenderStream(This,pCategory,pType,pSource,pfCompressor,pfRenderer) \
    (This)->lpVtbl->RenderStream(This,pCategory,pType,pSource,pfCompressor,pfRenderer)

#define ICaptureGraphBuilder2_ControlStream(This,pCategory,pType,pFilter,pstart,pstop,wStartCookie,wStopCookie) \
    (This)->lpVtbl->ControlStream(This,pCategory,pType,pFilter,pstart,pstop,wStartCookie,wStopCookie)

#define ICaptureGraphBuilder2_AllocCapFile(This,lpstr,dwSize) \
    (This)->lpVtbl->AllocCapFile(This,lpstr,dwSize)

#define ICaptureGraphBuilder2_CopyCaptureFile(This,lpwstrOld,lpwstrNew,fAllowEscAbort,pCallback) \
    (This)->lpVtbl->CopyCaptureFile(This,lpwstrOld,lpwstrNew,fAllowEscAbort,pCallback)

```

```

#define
ICaptureGraphBuilder2_FindPin(This,pSource,pindir,pCategory,pType,fUnco
nnected,num,ppPin) \
    (This)->lpVtbl ->
FindPin(This,pSource,pindir,pCategory,pType,fUnconnected,num,ppPin)

#endif /* COBJMACROS */

#endif /* C style interface */

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_SetFiltergraph_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ IGraphBuilder *pfg);

void __RPC_STUB ICaptureGraphBuilder2_SetFiltergraph_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_GetFiltergraph_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [out] */ IGraphBuilder **ppfg);

void __RPC_STUB ICaptureGraphBuilder2_GetFiltergraph_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

HRESULT STDMETHODCALLTYPE
ICaptureGraphBuilder2_SetOutputFileName_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pType,
    /* [in] */ LPCOLESTR lpstrFile,
    /* [out] */ IBaseFilter **ppf,
    /* [out] */ IFileSinkFilter **ppSink);

void __RPC_STUB ICaptureGraphBuilder2_SetOutputFileName_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [call_as] */ HRESULT STDMETHODCALLTYPE
ICaptureGraphBuilder2_RemoteFindInterface_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,

```

```

/* [in] */ const GUID *pType,
/* [in] */ IBaseFilter *pf,
/* [in] */ REFIID riid,
/* [out] */ IUnknown **ppint);

```

```

void __RPC_STUB ICaptureGraphBuilder2_RemoteFindInterface_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

```

```

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_RenderStream_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ IUnknown *pSource,
    /* [in] */ IBaseFilter *pfCompressor,
    /* [in] */ IBaseFilter *pfRenderer);

```

```

void __RPC_STUB ICaptureGraphBuilder2_RenderStream_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

```

```

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_ControlStream_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ IBaseFilter *pFilter,
    /* [in] */ REFERENCE_TIME *pstart,
    /* [in] */ REFERENCE_TIME *pstop,
    /* [in] */ WORD wStartCookie,
    /* [in] */ WORD wStopCookie);

```

```

void __RPC_STUB ICaptureGraphBuilder2_ControlStream_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

```

```

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_AllocCapFile_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ LPCOLESTR lpstr,
    /* [in] */ DWORDLONG dwlSize);

```

```

void __RPC_STUB ICaptureGraphBuilder2_AllocCapFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,

```

```

        DWORD *_pdwStubPhase);

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_CopyCaptureFile_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ LPOLESTR lpwstrOld,
    /* [in] */ LPOLESTR lpwstrNew,
    /* [in] */ int fAllowEscAbort,
    /* [in] */ IAMCopyCaptureFileProgress *pCallback);

void __RPC_STUB ICaptureGraphBuilder2_CopyCaptureFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

HRESULT STDMETHODCALLTYPE ICaptureGraphBuilder2_FindPin_Proxy(
    ICaptureGraphBuilder2 * This,
    /* [in] */ IUnknown *pSource,
    /* [in] */ PIN_DIRECTION pindir,
    /* [in] */ const GUID *pCategory,
    /* [in] */ const GUID *pType,
    /* [in] */ BOOL fUnconnected,
    /* [in] */ int num,
    /* [out] */ IPin **ppPin);

void __RPC_STUB ICaptureGraphBuilder2_FindPin_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

#endif      /* __ICaptureGraphBuilder2_INTERFACE_DEFINED__ */

#ifndef OUR_GUID_ENTRY
#define OUR_GUID_ENTRY(name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8) \
    DEFINE_GUID(name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8);
#endif

// BF87B6E1-8C27-11d0-B3F0-00AA003761C5      New Capture graph building
OUR_GUID_ENTRY(CLSID_CaptureGraphBuilder2,
0xBF87B6E1, 0x8C27, 0x11d0, 0xB3, 0xF0, 0x0, 0xAA, 0x00, 0x37, 0x61,
0xC5)

#if
!defined(AFX_EMAILDLG_H__C11DAB1E_416B_44D7_9E72_BED2CD614657__INCLUDED_)
)
#define AFX_EMAILDLG_H__C11DAB1E_416B_44D7_9E72_BED2CD614657__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

```

10

```

//
// Error Management.h
//
//
// Copyright (C) BKLK 2002.
// All rights reserved.
//
//
// DESCRIPTION
// General routines for managing errors
//
//
// Change Log:
// 11-Mar-02 (lck) - Created
//
////////////////////////////////////
////////

#ifndef __ERRORMANAGEMENT_H_
#define __ERRORMANAGEMENT_H_

enum {
    NMESSAGE_TYPE_ERROR = 1,
    NMESSAGE_TYPE_STATUS = 2,
    NMESSAGE_TYPE_DEBUG = 3
};

enum {
    NMESSAGE = WM_USER + 613
};

bool NProcessMessage(long lLineNumber, LPSTR szFilename, LPSTR
szMessage, int iMessageType);

#endif // __ERRORMANAGEMENT_H_
#if
!defined(AFX_EVENTDIALOG_H__8578BCA6_D7F7_4D4C_AC05_9AE9561D74B4__INCLU
DED_)
#define
AFX_EVENTDIALOG_H__8578BCA6_D7F7_4D4C_AC05_9AE9561D74B4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// EventDialog.h : header file
//

////////////////////////////////////
////////
// CEventDialog dialog

class CEventDialog : public CDialog
{
// Construction
public:
    CEventDialog(CWnd* pParent = NULL); // standard constructor

```

```

CImageList m_LargeImageList;
CImageList m_SmallImageList;
CImageList m_StateImageList;

CEventInfo m_EventInfo;
bool m_bNewEvent;
CString m_ConnectionLabel;

void      AddEmailAction();
void      AddPhoneAction();
void      AddVideoRecordAction();
void      AddX10Action();
int        GetSelectedItem();
DWORD     GetViewType();
BOOL      SetViewType(DWORD dwViewType);
bool      UpdateListCtrl();

// Dialog Data
//{{AFX_DATA(CEventDialog)
enum { IDD = IDD_EVENTDLG };
CEdit m_LabelCtrl;
CDateTimeCtrl m_EndTimeCtrl;
CDateTimeCtrl m_StartTimeCtrl;
CComboBox m_ListViewStyle;
CComboBox m_TypesOfActions;
CListCtrl m_ActionList;
CString m_Label;
BOOL m_Friday;
CString m_LabelTitle;
BOOL m_Monday;
BOOL m_Saturday;
CTime m_StartTime;
CString m_StartTimeTitle;
BOOL m_Sunday;
BOOL m_Thursday;
BOOL m_Tuesday;
BOOL m_Wednesday;
CTime m_EndTime;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CEventDialog)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CEventDialog)
virtual BOOL OnInitDialog();
afx_msg void OnAdd();

```

```

virtual void OnOK();
afx_msg void OnSelchangeListviewstyle();
afx_msg void OnEdit();
afx_msg void OnDelete();
afx_msg void OnDblclkActionlist(NMHDR* pNMHDR, LRESULT* pResult);
virtual void OnCancel();
afx_msg void OnSelectCleardays();
afx_msg void OnSelectFullweek();
afx_msg void OnSelectWeekdays();
afx_msg void OnSelectWeekend();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_EVENTDIALOG_H__8578BCA6_D7F7_4D4C_AC05_9AE9561D74B4__INCLU
DED_
#define AFX_EVENTDIALOG_H__8578BCA6_D7F7_4D4C_AC05_9AE9561D74B4__INCLU
DED_
#endif

#ifndef AFX_FEEDBACKERRORSVIEW_H__87BBC8DF_3385_4655_A3A0_C5B886E1ABAC
__INCLUDED_
#define AFX_FEEDBACKERRORSVIEW_H__87BBC8DF_3385_4655_A3A0_C5B886E1ABAC
__INCLUDED_
#endif

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeedbackErrorsView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CFeedbackErrorsView view

class CFeedbackErrorsView : public CEditView
{
protected:
    CFeedbackErrorsView();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CFeedbackErrorsView)

// Attributes
public:
    CString m_strText;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFeedbackErrorsView)
protected:

```

```

        virtual void OnDraw(CDC* pDC);        // overridden to draw this
view
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
        ///}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CFeedbackErrorsView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        ///{AFX_MSG(CFeedbackErrorsView)
        // NOTE - the ClassWizard will add and remove member
functions here.
        ///}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////

///{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_FEEDBACKERRORSVIEW_H__87BBC8DF_3385_4655_A3A0_C5B886E1ABAC
_INCLUDED_
#define AFX_FEEDBACKERRORSVIEW_H__87BBC8DF_3385_4655_A3A0_C5B886E1ABAC
_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeedbackFrame.h : header file
//

////////////////////////////////////
////////
// CFeedbackFrame frame

class CFeedbackFrame : public CMDIChildWnd
{
        DECLARE_DYNCREATE(CFeedbackFrame)
protected:
        CFeedbackFrame();        // protected constructor used by
dynamic creation

// Attributes

```

```

protected:
    CSplitterWnd m_wndSplitter;
    CToolBar      m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CFeedbackFrame)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CFeedbackFrame();

    // Generated message map functions
    //{AFX_MSG(CFeedbackFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg void OnClose();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_FEEDBACKFRAME_H__4EFDE4FC_E8AD_4BA5_978A_D192855C6B34__INC
LUDED_
#endif
#ifndef AFX_FEEDBACKINFODLG_H__DDAAA3DF_0416_4A47_81CC_F7338302121D__I
NCLUDED_
#define
AFX_FEEDBACKINFODLG_H__DDAAA3DF_0416_4A47_81CC_F7338302121D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeedbackInfoDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CFeedbackInfoDlg dialog

class CFeedbackInfoDlg : public CDialog
{
// Construction
public:

```

```

        CFeedbackInfoDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
//{{AFX_DATA(CFeedbackInfoDlg)
enum { IDD = IDD_FEEDBACKDLG };
CString      m_Description;
CString      m_Location;
CString      m_Message;
CString      m_Time;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CFeedbackInfoDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    {{{AFX_MSG(CFeedbackInfoDlg)
        // NOTE: the ClassWizard will add member functions here
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_FEEDBACKINFODLG_H__DDAAA3DF_0416_4A47_81CC_F7338302121D__I
NCLUDED_)
#if
!defined(AFX_FEEDBACKLISTVIEW_H__D9DD1740_21EA_4B64_8B86_B69CBB7BEB2C__
INCLUDED_)
#define
AFX_FEEDBACKLISTVIEW_H__D9DD1740_21EA_4B64_8B86_B69CBB7BEB2C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeedbackListView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CFeedbackListView view

class CFeedbackListView : public CListView
{
protected:

```

```

        CFeedbackListView();          // protected constructor used by
dynamic creation
        DECLARE_DYNCREATE(CFeedbackListView)

// Attributes
public:
        CImageList m_LargeImageList;
        CImageList m_SmallImageList;
        CImageList m_StateImageList;

// Operations
public:
        void          AddMessage(long lLineNumber, LPSTR szFilename, int
nType, CString strMessage, CString strDescription);
        int           GetSelectedItem();
        DWORD          GetViewType();
        BOOL            SetColumnWidth(int Column, int Width);
        BOOL            SetViewType(DWORD dwViewType);

// Overrides
        // ClassWizard generated virtual function overrides
        //{AFX_VIRTUAL(CFeedbackListView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CFeedbackListView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{AFX_MSG(CFeedbackListView)
        afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnFeedbackDebug();
        afx_msg void OnUpdateFeedbackDebug(CCmdUI* pCmdUI);
        afx_msg void OnFeedbackError();
        afx_msg void OnUpdateFeedbackError(CCmdUI* pCmdUI);
        afx_msg void OnFeedbackStatus();
        afx_msg void OnUpdateFeedbackStatus(CCmdUI* pCmdUI);
        afx_msg void OnFeedbackWarning();
        afx_msg void OnUpdateFeedbackWarning(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_FEEDBACKLISTVIEW_H_D9DD1740_21EA_4B64_8B86_B69CBB7BEB2C__
INCLUDED_)
#if
!defined(AFX_FEEDBACKSTATUSVIEW_H_594A57D6_8695_4FDF_B1D6_E30017F7C828
__INCLUDED_)
#define
AFX_FEEDBACKSTATUSVIEW_H_594A57D6_8695_4FDF_B1D6_E30017F7C828__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeedbackStatusView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CFeedbackStatusView view

class CFeedbackStatusView : public CEditView
{
protected:
    CFeedbackStatusView();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CFeedbackStatusView)

// Attributes
public:
    CString m_strText;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFeedbackStatusView)
protected:
    virtual void OnDraw(CDC* pDC);    // overridden to draw this
view
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CFeedbackStatusView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions

```

```

protected:
    //{AFX_MSG(CFeedbackStatusView)
        // NOTE - the ClassWizard will add and remove member
functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_FEEDBACKSTATUSVIEW_H__594A57D6_8695_4FDF_B1D6_E30017F7C828
__INCLUDED_
#if
#ifndef AFX_INSTANTMESSENGERCONVERSATIONVIEW_H__066841DA_F9AB_4558_A86
0_FA8D79867E94__INCLUDED_
#define
AFX_INSTANTMESSENGERCONVERSATIONVIEW_H__066841DA_F9AB_4558_A860_FA8D798
67E94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InstantMessengerConversationView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
// CInstantMessengerConversationView view

class CInstantMessengerConversationView : public CListView
{
protected:
    CInstantMessengerConversationView();          // protected
constructor used by dynamic creation
    DECLARE_DYNCREATE(CInstantMessengerConversationView)

// Attributes
public:
    CImageList m_LargeImageList;
    CImageList m_SmallImageList;
    CImageList m_StateImageList;

// Operations
public:
    BOOL          SetColumnWidth(int Column, int Width);

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CInstantMessengerConversationView)
public:
    virtual void OnInitialUpdate();

```

```

        protected:
        virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CInstantMessengerConversationView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CInstantMessengerConversationView)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_INSTANTMESSENGERCONVERSATIONVIEW_H__066841DA_F9AB_4558_A86
0_FA8D79867E94__INCLUDED_
#endif
#ifdef AFX_INSTANTMESSENGERDOC_H__9C1EFEDD_3E27_4AB9_81DB_4E0AFE9BCF
B__INCLUDED_
#define
AFX_INSTANTMESSENGERDOC_H__9C1EFEDD_3E27_4AB9_81DB_4E0AFE9BCFB__INCLUD
ED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InstantMessengerDoc.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////
// CInstantMessengerDoc document

class CInstantMessengerDoc : public CDocument
{
protected:
        CInstantMessengerDoc();          // protected constructor used
by dynamic creation
        DECLARE_DYNCREATE(CInstantMessengerDoc)

```

```

// Attributes
public:
    CString          m_ConnectionLabel;

// Operations
public:
    BOOL            IsModified();

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CInstantMessengerDoc)
    public:
        virtual void Serialize(CArchive& ar);    // overridden for
document i/o
        virtual BOOL CanCloseFrame(CFrameWnd* pFrame);
    protected:
        virtual BOOL OnNewDocument();
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CInstantMessengerDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(CInstantMessengerDoc)
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_INSTANTMESSENGERDOC_H__9C1EFEDD_3E27_4AB9_81DB_4E0AFE9BCF
B__INCLUDED_
#endif
#ifdef AFX_INSTANTMESSENGERENTRYVIEW_H__40F4B35A_71CC_4FC2_86FC_AF828
9EBBA35__INCLUDED_
#define
AFX_INSTANTMESSENGERENTRYVIEW_H__40F4B35A_71CC_4FC2_86FC_AF8289EBBA35__
INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InstantMessengerEntryView.h : header file
//

////////////////////////////////////
////////

```

```

// CInstantMessengerEntryView view

class CInstantMessengerEntryView : public CEditView
{
protected:
    CInstantMessengerEntryView();          // protected constructor
used by dynamic creation
    DECLARE_DYNCREATE(CInstantMessengerEntryView)

// Attributes
public:
    bool          GetConnectionLabel(CString& strConnectionLabel);

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInstantMessengerEntryView)
protected:
    virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CInstantMessengerEntryView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{{AFX_MSG(CInstantMessengerEntryView)
    afx_msg void OnConnectionConnect();
    afx_msg void OnConnectionDisconnect();
    afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_INSTANTMESSENGERENTRYVIEW_H__40F4B35A_71CC_4FC2_86FC_AF828
9EBBA35__INCLUDED_
#if
#ifndef AFX_INSTANTMESSENGERFRAME_H__0C5C7697_8688_4F91_B031_2C65DA868
0D0__INCLUDED_

```

```

#define
AFX_INSTANTMESSENGERFRAME_H__0C5C7697_8688_4F91_B031_2C65DA8680D0__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InstantMessengerFrame.h : header file
//

/////////////////////////////////////////////////////////////////
/////
// CInstantMessengerFrame frame

class CInstantMessengerFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CInstantMessengerFrame)
protected:
    CInstantMessengerFrame();           // protected constructor used
by dynamic creation

// Attributes
public:
    CSplitterWnd m_wndSplitter;
    CToolBar      m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CInstantMessengerFrame)
    protected:
        virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CInstantMessengerFrame();

    // Generated message map functions
    //{AFX_MSG(CInstantMessengerFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
/////

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
!defined(AFX_INSTANTMESSENGERFRAME_H__0C5C7697_8688_4F91_B031_2C65DA868
ODO__INCLUDED_)
#if
!defined(AFX_ITEMLISTDLG_H__68B684F8_F2A4_4ABA_9699_2D6DCCFAD696__INCLU
DED_)
#define
AFX_ITEMLISTDLG_H__68B684F8_F2A4_4ABA_9699_2D6DCCFAD696__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ItemListDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CItemListDlg dialog

class CItemListDlg : public CDialog
{
// Construction
public:
    CItemListDlg(CWnd* pParent = NULL);    // standard constructor

    CStringArray* m_parrItems;

// Dialog Data
   //{{AFX_DATA(CItemListDlg)
    enum { IDD = IDD_ITEMLISTDLG };
    CListBox     m_ItemListCtrl;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CItemListDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CItemListDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
#ifndef AFX_ITEMLISTDLG_H__68B684F8_F2A4_4ABA_9699_2D6DCCFAD696__INCLU
DED_
#endif
#ifndef AFX_LOCALVIDEODOC_H__1624B117_8AC1_47E9_959B_C242B0B51075__INC
LUDED_
#define
AFX_LOCALVIDEODOC_H__1624B117_8AC1_47E9_959B_C242B0B51075__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LocalVideoDoc.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CLocalVideoDoc document

class CLocalVideoDoc : public CDocument
{
protected:
    CLocalVideoDoc();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CLocalVideoDoc)

// Attributes
public:
    CString          m_ConnectionLabel;
    HWND             m_hWndVideoView;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CLocalVideoDoc)
    public:
        virtual void Serialize(CArchive& ar);    // overridden for
document i/o
        virtual void OnCloseDocument();
        virtual BOOL CanCloseFrame(CFrameWnd* pFrame);
    protected:
        virtual BOOL OnNewDocument();
    //{AFX_VIRTUAL

// Implementation
public:
    virtual ~CLocalVideoDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(CLocalVideoDoc)

```

```

        // NOTE - the ClassWizard will add and remove member
        functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    };

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_LOCALVIDEODOC_H__1624B117_8AC1_47E9_959B_C242B0B51075__INC
    LUED_
    #if
    #ifndef AFX_LOCALVIDEOEVENTSVIEW_H__4EDDD653_0F71_4AB7_AEF6_E3455C2F09
    2E__INCLUDED_
    #define
    AFX_LOCALVIDEOEVENTSVIEW_H__4EDDD653_0F71_4AB7_AEF6_E3455C2F092E__INCLU
    DED_

    #if _MSC_VER > 1000
    #pragma once
    #endif // _MSC_VER > 1000
    // LocalVideoEventsView.h : header file
    //

    //////////////////////////////////////
    // CLocalVideoEventsView view

    class CLocalVideoEventsView : public CListView
    {
    protected:
        CLocalVideoEventsView();          // protected constructor used
        by dynamic creation
        DECLARE_DYNCREATE(CLocalVideoEventsView)

    // Attributes
    public:
        CImageList m_LargeImageList;
        CImageList m_SmallImageList;
        CImageList m_StateImageList;

    // Operations
    public:
        void          OnEditEvent() ;
        bool          GetConnectionLabel(CString& strConnectionLabel);
        bool          GetIndexFromLabel(int& t_Index) ;
        int           GetSelectedEventItem();
        bool          GetSelectedEventLabel(CString& strLabel) ;
        DWORD         GetViewType();
        bool          ReceiveEventsArray(CConnectionMsg* pConnectionMsg);
        bool          SendEventsArray();
        BOOL          SetColumnWidth(int Column, int Width);
        BOOL          SetViewType(DWORD dwViewType);
        bool          UpdateListCtrl();

```

```

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CLocalVideoEventsView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
//}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CLocalVideoEventsView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
//{{AFX_MSG(CLocalVideoEventsView)
afx_msg void OnConnectionNewAlarm();
afx_msg void OnConnectionNewEvent();
afx_msg void OnConnectionConnect();
afx_msg void OnConnectionDisconnect();
afx_msg void OnViewLargeicons();
afx_msg void OnViewList();
afx_msg void OnViewSmallicons();
afx_msg void OnViewDetails();
afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnConnectionDeleteEvent();
afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
afx_msg void OnLocalvideoPause();
afx_msg void OnLocalvideoPlay();
afx_msg void OnLocalvideoStop();
afx_msg void OnLocalvideoRecord();
afx_msg void OnUpdateLocalvideoPause(CCmdUI* pCmdUI);
afx_msg void OnUpdateLocalvideoRecord(CCmdUI* pCmdUI);
afx_msg void OnUpdateLocalvideoPlay(CCmdUI* pCmdUI);

afx_msg void OnUpdateLocalvideoStop(CCmdUI* pCmdUI);
afx_msg void OnLocalvideoMotiondetection();
afx_msg void OnUpdateConnectionConnect(CCmdUI* pCmdUI);
afx_msg void OnUpdateConnectionDisconnect(CCmdUI* pCmdUI);
afx_msg void OnDestroy();
afx_msg void OnUpdateConnectionDeleteEvent(CCmdUI* pCmdUI);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_LOCALVIDEOEVENTSVIEW_H__4EDDD653_0F71_4AB7_AEF6_E3455C2F09
2E__INCLUDED_)
#if
!defined(AFX_LOCALVIDEOFRAME_H__E9952567_D385_4584_B873_58602C503D6F__I
NCLUDED_)
#define
AFX_LOCALVIDEOFRAME_H__E9952567_D385_4584_B873_58602C503D6F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LocalVideoFrame.h : header file
//

////////////////////////////////////
/////
// CLocalVideoFrame frame

class CLocalVideoFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CLocalVideoFrame)
protected:
    CLocalVideoFrame();           // protected constructor used by
dynamic creation

// Attributes
public:
    CSplitterWnd m_wndSplitter;
    CToolBar      m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLocalVideoFrame)
protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CLocalVideoFrame();

    // Generated message map functions
    //{{AFX_MSG(CLocalVideoFrame)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_LOCALVIDEOFRAME_H__E9952567_D385_4584_B873_58602C503D6F__I
NCLUDED_
#endif
#ifndef AFX_LOCALVIDEOVIDEOVIEW_H__BDF2DC54_C4CC_470E_AC8A_57C7546150E
0__INCLUDED_
#define
AFX_LOCALVIDEOVIDEOVIEW_H__BDF2DC54_C4CC_470E_AC8A_57C7546150E0__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LocalVideoVideoView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CLocalVideoVideoView view

class CLocalVideoVideoView : public CView
{
protected:
    CLocalVideoVideoView();          // protected constructor used
by dynamic creation
    DECLARE_DYNCREATE(CLocalVideoVideoView)

// Attributes
public:
    UINT          m_nTimer;

// Operations
public:
    LRESULT        OnGraphNotify( WPARAM wParam, LPARAM lParam );
    bool           GetIndexFromLabel(int& t_Index) ;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLocalVideoVideoView)
    public:
        virtual void OnInitialUpdate();
    protected:
        virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    //}}AFX_VIRTUAL

// Implementation

```

```
protected:  
    virtual ~CLocalVideoVideoView();  
#ifdef _DEBUG  
    virtual void AssertValid() const;  
    virtual void Dump(CDumpContext& dc) const;  
#endif  
  
    // Generated message map functions  
protected:  
    //{AFX_MSG(CLocalVideoVideoView)  
afx_msg void OnLocalvideoPlay();  
afx_msg void OnLocalvideoStop();  
afx_msg void onViewDetails();  
afx_msg void onViewLargeicons();  
afx_msg void onViewList();  
afx_msg void onViewSmallicons();  
  
afx_msg void OnConnectionConnect();  
afx_msg void OnConnectionDisconnect();  
afx_msg void OnConnectionNewAlarm();  
afx_msg void OnConnectionNewEvent();  
afx_msg void OnConnectionDeleteEvent();  
afx_msg void OnLocalvideoPause();  
afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);  
afx_msg void OnMove(int x, int y);  
afx_msg void OnSize(UINT nType, int cx, int cy);  
afx_msg void OnLocalvideoRecord();  
afx_msg void OnUpdateLocalvideoPause(CCmndUI* pCmdUI);  
afx_msg void OnUpdateLocalvideoPlay(CCmndUI* pCmdUI);  
afx_msg void OnUpdateLocalvideoRecord(CCmndUI* pCmdUI);  
afx_msg void OnUpdateLocalvideoStop(CCmndUI* pCmdUI);  
afx_msg void OnLocalvideoMotiondetection();  
afx_msg void OnUpdateConnectionConnect(CCmndUI* pCmdUI);  
afx_msg void OnUpdateConnectionDisconnect(CCmndUI* pCmdUI);  
afx_msg void OnDestroy();  
afx_msg void OnTimer(UINT nIDEvent);  
//}}AFX_MSG  
DECLARE_MESSAGE_MAP()  
};  
  
/////////////////////////////////////  
/////
```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

```
#endif //  
#ifndef AFX_LOCALVIDEOVIDEOVIEW_H_BDF2DC54_C4CC_470E_AC8A_57C7546150E  
0_INCLUDED_  
// Main Doc.h : interface of the CMainDoc class  
//  
/////////////////////////////////////  
/////
```

```

#if
!defined(AFX_MAINDOC_H__AFFC572F_F73E_4163_8195_F05519E8538F__INCLUDED_)
)
#define AFX_MAINDOC_H__AFFC572F_F73E_4163_8195_F05519E8538F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainDoc : public CDocument
{
protected: // create from serialization only
    CMainDoc();
    DECLARE_DYNCREATE(CMainDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CMainDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
    //{AFX_MSG(CMainDoc)
    // NOTE - the ClassWizard will add and remove member
    functions here.
    // DO NOT EDIT what you see in these blocks of generated
    code !
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
!defined(AFX_MAINDOC_H__AFFC572F_F73E_4163_8195_F05519E8538F__INCLUDED_
)
// Main View.h : interface of the CMainView class
//
////////////////////////////////////
/////

#if
!defined(AFX_MAINVIEW_H__5F576D89_DF37_4451_80F1_7B9C95FC283D__INCLUDED_
_)
#define AFX_MAINVIEW_H__5F576D89_DF37_4451_80F1_7B9C95FC283D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainView : public CView
{
protected: // create from serialization only
    CMainView();
    DECLARE_DYNCREATE(CMainView)

// Attributes
public:
    CMainDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CMainView)
    public:
        virtual void OnDraw(CDC* pDC); // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    }AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{AFX_MSG(CMainView)

```

```

        // NOTE - the ClassWizard will add and remove member
functions here.
        //      DO NOT EDIT what you see in these blocks of generated
code !
        ///}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in Main View.cpp
inline CMainDoc* CMainView::GetDocument()
{ return (CMainDoc*)m_pDocument; }
#endif

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef(AFX_MAINVIEW_H__5F576D89_DF37_4451_80F1_7B9C95FC283D__INCLUDED_
)
// MainFrm.h : interface of the CMainFrame class
//
////////////////////////////////////
////////////////////////////////////

#ifdef(AFX_MAINFRM_H__519AF4AE_E662_4C27_8B3E_AB24C105F5AE__INCLUDED_
)
#define AFX_MAINFRM_H__519AF4AE_E662_4C27_8B3E_AB24C105F5AE__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:
    UINT m_nTimer;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    ///}AFX_VIRTUAL(CMainFrame)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    ///}AFX_VIRTUAL

```

[illegible]

```

// CMediaPlayer2 wrapper class

class CMediaPlayer2 : public CWnd
{
protected:
    DECLARE_DYNCREATE(CMediaPlayer2)
public:
    CLSID const& GetClsid()
    {
        static CLSID const clsid
            = { 0x22d6f312, 0xb0f6, 0x11d0, { 0x94, 0xab, 0x0,
0x80, 0xc7, 0x4c, 0x7e, 0x95 } };
        return clsid;
    }
    virtual BOOL Create(LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect,

        CWnd* pParentWnd, UINT nID,
        CCreateContext* pContext = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID); }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect, CWnd* pParentWnd, UINT nID,
        CFile* pPersist = NULL, BOOL bStorage = FALSE,
        BSTR bstrLicKey = NULL)
    { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID,
        pPersist, bStorage, bstrLicKey); }

// Attributes
public:

// Operations
public:
    double GetCurrentPosition();
    void SetCurrentPosition(double newValue);
    double GetDuration();
    long GetImageSourceWidth();
    long GetImageSourceHeight();
    long GetMarkerCount();
    BOOL GetCanScan();
    BOOL GetCanSeek();
    BOOL GetCanSeekToMarkers();
    long GetCurrentMarker();
    void SetCurrentMarker(long nNewValue);
    CString GetFileName();
    void SetFileName(LPCTSTR lpszNewValue);
    CString GetSourceLink();
    DATE GetCreationDate();
    CString GetErrorCorrection();
    long GetBandwidth();
    long GetSourceProtocol();
    long GetReceivedPackets();
    long GetRecoveredPackets();
    long GetLostPackets();

```

```
long GetReceptionQuality();
long GetBufferingCount();
BOOL GetIsBroadcast();
long GetBufferingProgress();
CString GetChannelName();
CString GetChannelDescription();
CString GetChannelURL();
CString GetContactAddress();
CString GetContactPhone();
CString GetContactEmail();
double GetBufferingTime();
void SetBufferingTime(double newValue);
BOOL GetAutoStart();
void SetAutoStart(BOOL bNewValue);
BOOL GetAutoRewind();
void SetAutoRewind(BOOL bNewValue);
double GetRate();
void SetRate(double newValue);
BOOL GetSendKeyboardEvents();
void SetSendKeyboardEvents(BOOL bNewValue);
BOOL GetSendMouseClickEvents();
void SetSendMouseClickEvents(BOOL bNewValue);
BOOL GetSendMouseMoveEvents();
void SetSendMouseMoveEvents(BOOL bNewValue);
long GetPlayCount();
void SetPlayCount(long nNewValue);
BOOL GetClickToPlay();
void SetClickToPlay(BOOL bNewValue);
BOOL GetAllowScan();
void SetAllowScan(BOOL bNewValue);
BOOL GetEnableContextMenu();
void SetEnableContextMenu(BOOL bNewValue);
long GetCursorType();
void SetCursorType(long nNewValue);
long GetCodecCount();
BOOL GetAllowChangeDisplaySize();
void SetAllowChangeDisplaySize(BOOL bNewValue);
BOOL GetIsDurationValid();
long GetOpenState();
BOOL GetSendOpenStateChangeEvents();
void SetSendOpenStateChangeEvents(BOOL bNewValue);
BOOL GetSendWarningEvents();
void SetSendWarningEvents(BOOL bNewValue);
BOOL GetSendErrorEvents();
void SetSendErrorEvents(BOOL bNewValue);
long GetPlayState();
BOOL GetSendPlayStateChangeEvents();
void SetSendPlayStateChangeEvents(BOOL bNewValue);
long GetDisplaySize();
void SetDisplaySize(long nNewValue);
BOOL GetInvokeURLs();
void SetInvokeURLs(BOOL bNewValue);
CString GetBaseURL();
void SetBaseURL(LPCTSTR lpszNewValue);
CString GetDefaultFrame();
void SetDefaultFrame(LPCTSTR lpszNewValue);
BOOL GetHasError();
```

```

CString GetErrorDescription();
long GetErrorCode();
BOOL GetAnimationAtStart();
void SetAnimationAtStart(BOOL bNewValue);
BOOL GetTransparentAtStart();
void SetTransparentAtStart(BOOL bNewValue);
long GetVolume();
void SetVolume(long nNewValue);
long GetBalance();
void SetBalance(long nNewValue);
long GetReadyState();
double GetSelectionStart();
void SetSelectionStart(double newValue);
double GetSelectionEnd();
void SetSelectionEnd(double newValue);
BOOL GetShowDisplay();
void SetShowDisplay(BOOL bNewValue);
BOOL GetShowControls();
void SetShowControls(BOOL bNewValue);
BOOL GetShowPositionControls();
void SetShowPositionControls(BOOL bNewValue);
BOOL GetShowTracker();
void SetShowTracker(BOOL bNewValue);
BOOL GetEnablePositionControls();
void SetEnablePositionControls(BOOL bNewValue);
BOOL GetEnableTracker();
void SetEnableTracker(BOOL bNewValue);
BOOL GetEnabled();
void SetEnabled(BOOL bNewValue);
unsigned long GetDisplayForeColor();
void SetDisplayForeColor(unsigned long newValue);
unsigned long GetDisplayBackColor();
void SetDisplayBackColor(unsigned long newValue);
long GetDisplayMode();
void SetDisplayMode(long nNewValue);
BOOL GetVideoBorder3D();
void SetVideoBorder3D(BOOL bNewValue);
long GetVideoBorderWidth();
void SetVideoBorderWidth(long nNewValue);
unsigned long GetVideoBorderColor();
void SetVideoBorderColor(unsigned long newValue);
BOOL GetShowGotoBar();
void SetShowGotoBar(BOOL bNewValue);
BOOL GetShowStatusBar();
void SetShowStatusBar(BOOL bNewValue);
BOOL GetShowCaptioning();
void SetShowCaptioning(BOOL bNewValue);
BOOL GetShowAudioControls();
void SetShowAudioControls(BOOL bNewValue);
CString GetCaptioningID();
void SetCaptioningID(LPCTSTR lpszNewValue);
BOOL GetMute();
void SetMute(BOOL bNewValue);
BOOL GetCanPreview();
BOOL GetPreviewMode();
void SetPreviewMode(BOOL bNewValue);
BOOL GetHasMultipleItems();

```

```

long GetLanguage();
void SetLanguage(long nNewValue);
long GetAudioStr am();
void SetAudioStream(long nNewValue);
CString GetSAMISStyle();
void SetSAMISStyle(LPCTSTR lpszNewValue);
CString GetSAMILang();
void SetSAMILang(LPCTSTR lpszNewValue);
CString GetSAMIFileName();
void SetSAMIFileName(LPCTSTR lpszNewValue);
long GetStreamCount();
CString GetClientId();
long GetConnectionSpeed();
BOOL GetAutoSize();
void SetAutoSize(BOOL bNewValue);
BOOL GetEnableFullScreenControls();
void SetEnableFullScreenControls(BOOL bNewValue);
LPDISPATCH GetActiveMovie();
LPDISPATCH GetNSPlay();
BOOL GetWindowlessVideo();
void SetWindowlessVideo(BOOL bNewValue);
void Play();
void Stop();
void Pause();
double GetMarkerTime(long MarkerNum);
CString GetMarkerName(long MarkerNum);
void AboutBox();
BOOL GetCodecInstalled(long CodecNum);
CString GetCodecDescription(long CodecNum);
CString GetCodecURL(long CodecNum);
CString GetMoreInfoURL(long MoreInfoType);
CString GetMediaInfoString(long MediaInfoType);
void Cancel();
void Open(LPCTSTR bstrFileName);
BOOL IsSoundCardEnabled();
void Next();
void Previous();
void StreamSelect(long StreamNum);
void FastForward();
void FastReverse();
CString GetStreamName(long StreamNum);
long GetStreamGroup(long StreamNum);
BOOL GetStreamSelected(long StreamNum);
CMediaPlayerDvd GetDvd();
CString GetMediaParameter(long EntryNum, LPCTSTR
bstrParameterName);
CString GetMediaParameterName(long EntryNum, long Index);
long GetEntryCount();
long GetCurrentEntry();
void SetCurrentEntry(long EntryNumber);
void ShowDialog(long mpDialogIndex);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
#ifndef AFX_MEDIAPLAYER2_H__DE7C7306_D997_4CD4_8227_OEF1B9A482A8__INCL
    UDED_
#endif
#ifndef AFX_MEDIAPLAYERDVD_H__D8763923_C300_4B68_838D_ABA283ED2F0A__IN
    CLUDED_
#define
AFX_MEDIAPLAYERDVD_H__D8763923_C300_4B68_838D_ABA283ED2F0A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
// regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

////////////////////////////////////
// CMediaPlayerDvd wrapper class

class CMediaPlayerDvd : public COleDispatchDriver
{
public:
    CMediaPlayerDvd() {} // Calls COleDispatchDriver default
    constructor
    CMediaPlayerDvd(LPDISPATCH pDispatch) :
    COleDispatchDriver(pDispatch) {}
    CMediaPlayerDvd(const CMediaPlayerDvd& dispatchSrc) :
    COleDispatchDriver(dispatchSrc) {}

// Attributes
public:

// Operations
public:
    void ButtonSelectAndActivate(unsigned long uiButton);
    void UpperButtonSelect();
    void LowerButtonSelect();
    void LeftButtonSelect();
    void RightButtonSelect();
    void ButtonActivate();
    void ForwardScan(double dwSpeed);
    void BackwardScan(double dwSpeed);
    void PrevPGSearch();
    void TopPGSearch();
    void NextPGSearch();
    void TitlePlay(unsigned long uiTitle);
    void ChapterPlay(unsigned long uiTitle, unsigned long uiChapter);
    void ChapterSearch(unsigned long Chapter);
    void MenuCall(long MenuID);
    void ResumeFromMenu();
    void TimePlay(unsigned long uiTitle, LPCTSTR bstrTime);
    void TimeSearch(LPCTSTR bstrTime);

```

```

    void ChapterPlayAutoStop(unsigned long ulTitle, unsigned long
ulChapter, unsigned long ulChaptersToPlay);
    void StillOff();
    void GoUp();
    CString GetTotalTitleTime();
    unsigned long GetNumberOfChapters(unsigned long ulTitle);
    CString GetAudioLanguage(unsigned long ulStream);
    CString GetSubpictureLanguage(unsigned long ulStream);
    VARIANT GetAllGPRMs();
    VARIANT GetAllSPRMs();
    BOOL UOPValid(unsigned long ulUOP);
    unsigned long GetButtonsAvailable();
    unsigned long GetCurrentButton();
    unsigned long GetAudioStreamsAvailable();
    unsigned long GetCurrentAudioStream();
    void SetCurrentAudioStream(unsigned long newValue);
    unsigned long GetCurrentSubpictureStream();
    void SetCurrentSubpictureStream(unsigned long newValue);
    unsigned long GetSubpictureStreamsAvailable();
    BOOL GetSubpictureOn();
    void SetSubpictureOn(BOOL bNewValue);
    unsigned long GetAnglesAvailable();
    unsigned long GetCurrentAngle();
    void SetCurrentAngle(unsigned long newValue);
    unsigned long GetCurrentTitle();
    unsigned long GetCurrentChapter();
    CString GetCurrentTime();
    void SetRoot(LPCTSTR lpszNewValue);
    CString GetRoot();
    unsigned long GetFramesPerSecond();
    unsigned long GetCurrentDomain();
    unsigned long GetTitlesAvailable();
    unsigned long GetVolumesAvailable();
    unsigned long GetCurrentVolume();
    unsigned long GetCurrentDiscSide();
    BOOL GetCCActive();
    void SetCCActive(BOOL bNewValue);
    unsigned long GetCurrentCCService();
    void SetCurrentCCService(unsigned long newValue);
    CString GetUniqueID();
    unsigned long GetColorKey();
    void SetColorKey(unsigned long newValue);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MEDIAPLAYERDVD_H__D8763923_C300_4B68_838D_ABA283ED2F0A__IN
CLUDED_)

enum enum_NUPDATE
{
    NUPDATE_NEWCONNECTION = 1,
    NUPDATE_DELETECONNECTION,
    NUPDATE_PREDELETECONNECTION,

```

```

    NUPDATE_UPDATECONNECTION,
    NUPDATE_ERRORMESSAGE,
    NUPDATE_STATUSMESSAGE,
    NUPDATE_DEBUGMESSAGE,
    NUPDATE_DISCONNECTCONNECTION,
    NUPDATE_CLOSECONNECTIONWINDOW,
    NUPDATE_DPMESSAGERECEIVED,
    NUPDATE_IMMESSAGESENT,
    NUPDATE_OPENPROFILE,
    NUPDATE_REFRESH
};

// #define NPROCESSSHRMESSAGE(hResult, nMessageType, szFunctionName) {
NProcessMessage(__LINE__, __FILE__, (HRESULT) hResult, nMessageType,
szFunctionName); }
// #define NPROCESSIDSMESSAGE(wStringID, nMessageType, szFunctionName)
{ NProcessMessage(__LINE__, __FILE__, (WORD) wStringID, nMessageType,
szFunctionName); }

bool NProcessMessage(long lLineNumber, LPSTR szFilename, HRESULT
hResult, int nMessageType, LPSTR szFunctionName = NULL);
bool NProcessMessage(long lLineNumber, LPSTR szFilename, WORD
wStringID, int nMessageType, LPSTR szFunctionName = NULL);

void NUpdateAllViews( CView* pSender, LPARAM lHint = 0L, CObject* pHint
= NULL );

#ifdef AFX_MOTIONDETECTIONSETTINGSDDL_H_F3DC5BB3_AA95_46C5_9338_DF0D
0F483265_INCLUDED_
#define
AFX_MOTIONDETECTIONSETTINGSDDL_H_F3DC5BB3_AA95_46C5_9338_DF0D0F483265_
_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MotionDetectionSettingsDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CMotionDetectionSettingsDlg dialog

class CMotionDetectionSettingsDlg : public CDialog
{
// Construction
public:
    CMotionDetectionSettingsDlg(CWnd* pParent = NULL);    // standard
    constructor

// Dialog Data
    #ifdef AFX_DATA(CMotionDetectionSettingsDlg)
    enum { IDD = IDD_MDSETTINGSDDL };
    CSliderCtrl m_SensitivityCtrl;

```

```

    CEdit m_DwellTimeCtrl;
    BOOL m_Active;
    UINT m_DwellTime;
    int m_Sensitivity;
    //}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMotionDetectionSettingsDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CMotionDetectionSettingsDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_MOTIONDETECTIONSETTINGSDDL_H__F3DC5BB3_AA95_46C5_9338_DF0D_
0F483265__INCLUDED_
#define AFX_MOTIONDETECTIONSETTINGSDDL_H__F3DC5BB3_AA95_46C5_9338_DF0D_
0F483265__INCLUDED_
#endif

#ifdef AFX_PHONEDLG_H__26D81F07_2B27_4FF9_A9C1_A765E330FA36__INCLUDED_
#define AFX_PHONEDLG_H__26D81F07_2B27_4FF9_A9C1_A765E330FA36__INCLUDED_
#endif

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PhoneDlg.h : header file
//

////////////////////
////////
// CPhoneDlg dialog

class CPhoneDlg : public CDialog
{
// Construction
public:
    CPhoneDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{AFX_DATA(CPhoneDlg)
    enum { IDD = IDD_PHONE };
    CButton m_TestCtrl;

```

[illegible]

```

class CPlaybackDoc : public CDocument
{
protected:
    CPlaybackDoc();           // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CPlaybackDoc)

// Attributes
public:
    CString m_strVideoFile;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CPlaybackDoc)
    public:
    virtual void Serialize(CArchive& ar);    // overridden for
document i/o
    protected:
    virtual BOOL OnNewDocument();
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CPlaybackDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(CPlaybackDoc)
    // NOTE - the ClassWizard will add and remove member
functions here.
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_PLAYBACKDOC_H__B3956769_3587_46D2_80CD_A5C0194F0BBE__INCLU
DED_
#define AFX_PLAYBACKDOC_H__B3956769_3587_46D2_80CD_A5C0194F0BBE__INCLU
DED_
#endif

#ifndef AFX_PLAYBACKFRAME_H__E23E484F_5677_4285_9827_1642EB64EBAC__INC
LUDED_
#define AFX_PLAYBACKFRAME_H__E23E484F_5677_4285_9827_1642EB64EBAC__INC
LUDED_
#endif

#ifdef AFX_PLAYBACKFRAME_H__E23E484F_5677_4285_9827_1642EB64EBAC__INCLUDED_
#define AFX_PLAYBACKFRAME_H__E23E484F_5677_4285_9827_1642EB64EBAC__INCLUDED_
#endif

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

```

```

// PlaybackFrame.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackFrame frame

class CPlaybackFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CPlaybackFrame)
protected:
    CPlaybackFrame();          // protected constructor used by
dynamic creation

// Attributes
public:
    CSplitterWnd m_wndSplitter;
    CToolBar      m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CPlaybackFrame)
protected:
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //{AFX_VIRTUAL

// Implementation
protected:
    virtual ~CPlaybackFrame();

    // Generated message map functions
    //{AFX_MSG(CPlaybackFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    afx_msg void OnDestroy();
    //{AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////

//{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_PLAYBACKFRAME_H__E23E484F_5677_4285_9827_1642EB64EBAC__INC
LUDED_
#endif
#ifndef AFX_PLAYBACKLISTVIEW_H__BC0A3994_717E_4A98_8801_FA84506AC86B__
INCLUDED_

```

```

#define
AFX_PLAYBACKLISTVIEW_H__BC0A3994_717E_4A98_8801_FA84506AC86B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PlaybackListView.h : header file
//

/////////////////////////////////////////////////////////////////
/////
// CPlaybackListView view

class CPlaybackListView : public CListView
{
protected:
    CPlaybackListView();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CPlaybackListView)

// Attributes
public:
    CImageList m_LargeImageList;
    CImageList m_SmallImageList;
    CImageList m_StateImageList;

// Operations
public:
    bool        GetSelectedFilename(CString& strFilename) ;
    int         GetSelectedItem();
    bool        GetVideoPlaybackFiles(CStringArray& arrFiles);
    DWORD       GetViewType();
    bool        LoadVideoPlaybackFiles();
    BOOL        SetColumnWidth(int Column, int Width);
    BOOL        SetViewType(DWORD dwViewType);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPlaybackListView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CPlaybackListView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
}

```

```

        // Generated message map functions
protected:
    //{AFX_MSG(CPlaybackListView)
    afx_msg void OnPlaybackFastforward();
    afx_msg void OnPlaybackForward();
    afx_msg void OnPlaybackGo();
    afx_msg void OnPlaybackPause();
    afx_msg void OnPlaybackStop();
    afx_msg void OnViewDetails();
    afx_msg void OnViewLargeicons();
    afx_msg void OnViewList();
    afx_msg void OnViewSmallicons();
    afx_msg void OnOpenVideoFile();
    afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    afx_msg void OnPlaySelectedVideo();
    afx_msg void OnDeleteVideoFile();
    afx_msg void OnUpdateVideoPlayback();
    afx_msg void OnOpenVideoDirectory();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifdef(AFX_PLAYBACKLISTVIEW_H__BC0A3994_717E_4A98_8801_FA84506AC86B__
INCLUDED_)
#ifdef(AFX_PLAYBACKVIDEOVIEW_H__C0E07D26_5A39_44A1_A085_B5C05C38151F__
INCLUDED_)
#define
AFX_PLAYBACKVIDEOVIEW_H__C0E07D26_5A39_44A1_A085_B5C05C38151F__INCLUDED
-

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PlaybackVideoView.h : header file
//

#include "SimpleVideo.h"

////////////////////////////////////
////////////////////////////////////
// CPlaybackVideoView view

class CPlaybackVideoView : public CView
{
protected:
    CPlaybackVideoView();          // protected constructor used by
dynamic creation

```

```

        DECLARE_DYNCREATE(CPlaybackVideoView)

// Attributes
public:
    CSimpleVideo m_SimpleVideo;
    bool         PlaySelectedVideoFile();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CPlaybackVideoView)
    protected:
        virtual void OnDraw(CDC* pDC);          // overridden to draw this
view
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
        virtual LRESULT DefWindowProc(UINT message, WPARAM wParam, LPARAM
lParam);
    }AFX_VIRTUAL

// Implementation
protected:
    virtual ~CPlaybackVideoView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
    // Generated message map functions
protected:
    //{AFX_MSG(CPlaybackVideoView)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnOpenVideoFile();
    afx_msg void OnPlaybackForward();
    afx_msg void OnPlaybackGo();
    afx_msg void OnPlaybackPause();
    afx_msg void OnPlaybackStop();
    afx_msg void OnPlaybackFastforward();
    afx_msg void OnViewDetails();
    afx_msg void OnViewLargeicons();
    afx_msg void OnViewList();
    afx_msg void OnViewSmallicons();
    afx_msg void OnMove(int x, int y);
    afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    afx_msg void OnPlaySelectedVideo();
    afx_msg void OnDeleteVideoFile();
    }AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PLAYBACKVIDEOVIEW_H__C0E07D26_5A39_44A1_A085_B5C05C38151F_
_INCLUDED_)
//{{AFX_INCLUDES()}
#include "activemovie3.h"
#include "mediaplayer2.h"
//}}AFX_INCLUDES
#if
!defined(AFX_PLAYBACKWMPVIEW_H__FA3A3AED_2AC2_4A43_84F3_EB5CB8B48B0C__I
NCLUDED_)
#define
AFX_PLAYBACKWMPVIEW_H__FA3A3AED_2AC2_4A43_84F3_EB5CB8B48B0C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PlaybackWMPView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackWMPView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CPlaybackWMPView : public CFormView
{
protected:
    CPlaybackWMPView();          // protected constructor used by
dynamic creation
    DECLARE_DYNCREATE(CPlaybackWMPView)

// Form Data
public:
    //{{AFX_DATA(CPlaybackWMPView)
    enum { IDD = IDD_FORMVIEWVIDEOPLAYBACK };
    CMediaPlayer2      m_WMP;
    //}}AFX_DATA

// Attributes
public:
    bool m_bIsWMPInitialized;

// Operations
public:
    void ResizeWMP();
    bool PlaySelectedVideoFile();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPlaybackWMPView)

```

```

    public:
        virtual void OnInitialUpdat ();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
        //{AFX_VIRTUAL

// Implementation
protected:
        virtual ~CPlaybackWMPView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
        //{AFX_MSG(CPlaybackWMPView)
        afx_msg void OnMove(int x, int y);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnOpenVideoFile();
        //{AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
////////////////////////////////////

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_PLAYBACKWMPVIEW_H__FA3A3AED_2AC2_4A43_84F3_EB5CB8B48B0C__I
NCLUDED_
// Project Nalay.h : main header file for the PROJECT NALAY application
//

#ifdef AFX_PROJECTNALAY_H__BBDBDB4F_4B8B_423F_A4F1_7EBC917ADAAC__INCL
UDED_
#define
AFX_PROJECTNALAY_H__BBDBDB4F_4B8B_423F_A4F1_7EBC917ADAAC__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifdef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"    // main symbols

```

```

////////////////////////////////////
////////////////////////////////////
//
//      Connection Header Information
//
enum enum_CONNECTIONMESGTYPE
{
    CONNECTIONMESGTYPE_IMMESSAGE,
    CONNECTIONMESGTYPE_LOCALVIDEOEVENTS
};

#define CONNECTIONMESG_BUFFERLENGTH      1024

class CConnectionMsg : public CObject
{
public:
    int          m_nSize;
    bool         m_bLocal;
    int          m_nMessageType ;
    char         m_szMessage[128];
    char         m_szLabel[64];
    BYTE         m_pData[CONNECTIONMESG_BUFFERLENGTH];

public:
    CConnectionMsg();

    bool         SetLabel(CString strLabel);
    bool         SetMessage(CString strMessage);
};
//
////////////////////////////////////
////////////////////////////////////
//
//      FeedbackHeader Information
//
enum enum_FEEDBACKMESSAGETYPE
{
    FEEDBACKMESSAGETYPE_STATUS,
    FEEDBACKMESSAGETYPE_WARNING,
    FEEDBACKMESSAGETYPE_ERROR,
    FEEDBACKMESSAGETYPE_DEBUG
};

void SendFeedbackMessage(long lLineNumber, LPSTR szFilename, int nType,
    CString strMessage, CString strDescription);

void SendFeedbackHResult(long lLineNumber, LPSTR szFilename, int nType,
    CString strMessage, HRESULT hResult);
void SendFeedbackIDS(long lLineNumber, LPSTR szFilename, int nType,
    CString strMessage, UINT nStringID);
#define NSENDFEEDBACKMESSAGE(nType, strMessage, strDescription) {
    SendFeedbackMessage(__LINE__, __FILE__, nType, strMessage,
    strDescription); }

```

```

#define NSENDFEEDBACKHRESULT(nType, strMessage, hResult) {
SendFeedbackHResult(__LINE__, __FILE__, nType, strMessage, hResult); }
#define NSENDFEEDBACKIDS(nType, strMessage, nStringID) {
SendFeedbackIDS(__LINE__, __FILE__, nType, strMessage, nStringID); }

void SendIMessage(CConnectionMsg* pConnectionMsg);
void SendVSMMessage(DWORD dwMessageId, CConnectionMsg* pConnectionMsg =
NULL);

//
////////////////////////////////////
////////////////////////////////////

////////////////////////////////////
////////////////////////////////////
//
//      Update Header Information
//
enum enum_NUPDATE
{
    NUPDATE_NEWCONNECTION = 1,
    NUPDATE_DELETECONNECTION,
    NUPDATE_PREDELETECONNECTION,
    NUPDATE_UPDATECONNECTION,
    NUPDATE_CONNECT,
    NUPDATE_DISCONNECT,
    NUPDATE_CLOSECONNECTIONWINDOW,
    NUPDATE_DPMESSAGERECEIVED,
    NUPDATE_IMMESSAGERECEIVED,
    NUPDATE_IMMESSAGESENT,
    NUPDATE_OPENPROFILE,
    NUPDATE_REFRESH,
    NUPDATE_NEWVIDEOFILE,
    NUPDATE_PLAYSELECTEDVIDEOFILE,
    NUPDATE_OPENVIDEOFILE,
    NUPDATE_DELETEVIDEOFILE,
    NUPDATE_PLAYBACKFORWARD,
    NUPDATE_PLAYBACKGO,
    NUPDATE_PLAYBACKPAUSE,
    NUPDATE_PLAYBACKSTOP,
    NUPDATE_PLAYBACKFASTFORWARD,
    NUPDATE_VIEWDETAILS,
    NUPDATE_VIEWLARGEICONS,
    NUPDATE_VIEWLIST,
    NUPDATE_VIEWSMALLICONS,
    NUPDATE_NEWALARM,
    NUPDATE_NEWSCHEDULEDEVENT,
    NUPDATE_DELETEEVENT,
    NUPDATE_LOCALVIDEOPAUSE,
    NUPDATE_LOCALVIDEOPLAY,
    NUPDATE_LOCALVIDEOSTOP,
    NUPDATE_LOCALVIDEORECORD,
    NUPDATE_LOCALVIDEOMOTIONDETECTION,
};

void NUpdateAllViews( CView* pSender, LPARAM lHint = 0L, CObject* pHint
= NULL );

```

```

//
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
//    General purpose functions
//
bool PromptForFolder(HWND hWnd, char *promptText, char
*selectedFolderPath);
void DisplayContextMenu(CWnd* pWnd, CPoint point, UINT IDResource);
bool GetMostRecentVideoFile(CString& t_strVideoFile);
CString GetCurrentIPAddress();
bool PromptSecurityPassword(UINT AppSecurityLoc);

//
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

class CLayoutConnectionWindow : public CObject
{
public:
    CString          m_Label;
    CRect            m_rectWindow;

public:
    DECLARE_SERIAL( CLayoutConnectionWindow )
    CLayoutConnectionWindow();

    CLayoutConnectionWindow& operator=(CLayoutConnectionWindow&
LayoutConnectionWindow);
    void              Serialize( CArchive& archive );
};

class CLayout : CObject
{
public:
    // Main Window
    CRect            m_rectMainWindow;

    // Conections window
    BOOL             m_bConnectionsWindowOpen;
    CRect            m_rectConnectionsWindow;

    // Feedback window
    BOOL             m_bFeedbackWindowOpen;
    CRect            m_rectFeedbackWindow;

    // Video Playback window
    BOOL             m_bPlaybackWindowOpen;
    CRect            m_rectPlaybackWindow;

    // Connection Windows
    CArray<CLayoutConnectionWindow, CLayoutConnectionWindow&>
m_LayoutConnections;

```

```

public:
    DECLARE_SERIAL( CLayout )
    CLayout();

    bool        AddLayoutConnectionWindow(CString
strConnectionLabel);
    bool        DeleteLayoutConnection(CString strConnectionLabel);
    bool        GetConnectionWindowRect(CString strConnectionLabel,
CRect & RectWindow );
    int         GetLayoutConnectionIndexFromLabel(CString
strLabel);
    bool        GetSerializeFileName(CString& strFileName);
    bool        LoadSettings() ;
    void        SaveSettings() ;
    void        Serialize( CArchive& archive );
    bool        SetConnectionWindowRect(CString strConnectionLabel,
CRect RectWindow );
};

enum enum_ONDBLCLICKCONNECTIONITEM
{
    ONDBLCLICKCONNECTIONITEM_OPENCONNECTION,
    ONDBLCLICKCONNECTIONITEM_CONFIGCONNECTION
};

enum enum_PWDPROMPT
{
    PWDPROMPT_ONLAUNCH,
    PWDPROMPT_ONEXIT,
    PWDPROMPT_ONCONNECTION,
    PWDPROMPT_ONALARM,
    PWDPROMPT_ONCONFIGURATION,
};

class CAppConfig : CObject
{
public:
    // Startup location
    CString      m_StartupPath;

    // Connections
    UINT         m_nOnDoubleClickConnectionsItem;

    // Feedback
    BOOL         m_ShowDebugFeedback;
    BOOL         m_ShowErrorFeedback;
    BOOL         m_ShowStatusFeedback;
    BOOL         m_ShowWarningFeedback;

    // TAPI
    CString      m_TAPIDevice;

    // Video Record
    CTime        m_DefaultVideoRecordDuration;
    CString      m_DefaultVideoRecordFilename;
    int          m_MaxContinuousFiles;

```

```

// Video Playback
int          m_ForwardSpeed;
int          m_FastForwardSpeed;
CString      m_DefaultVideoPlaybackDirectory;

// Yellow Pages
BOOL         m_AutoRegisterYellowPages;

// Audio Play & Record
CString      m_DefaultAudioDirectory;

// Security Prompt
BOOL         m_PromptOnExit;
BOOL         m_EnableAppSecurityPrompt;
CString      m_AppPassword;
BOOL         m_PwdPromptOnLaunch;
BOOL         m_PwdPromptOnExit;
BOOL         m_PwdPromptOnAlarm;
BOOL         m_PwdPromptOnConnection;
BOOL         m_PwdPromptOnConfiguration;

// Update
BOOL         m_CheckUpdateAtLaunch;

public:
    DECLARE_SERIAL( CAppConfig )
    CAppConfig();

    bool      GetSerializeFileName(CString& strFileName);
    bool      LoadSettings() ;
    void      SaveSettings() ;
    void      Serialize( CArchive& archive );
};

////////////////////////////////////
/////
// CProjectNalayApp:
// See Project Nalay.cpp for the implementation of this class
//

class CProjectNalayApp : public CWinApp
{
public:
    CMultiDocTemplate* m_pDocTemplateLocalVideo;
    CMultiDocTemplate* m_pDocTemplateConnections;
    CMultiDocTemplate* m_pDocTemplateFeedback;
    CMultiDocTemplate* m_pDocTemplateRemoteVideo;
    CMultiDocTemplate* m_pDocTemplatePlayback;
    CMultiDocTemplate* m_pDocTemplateInstantMessenger;

    // Feedback view
    CFeedbackListView* m_pFeedbackListView;

public:
    CProjectNalayApp();

```

```

        void      NUpdat AllViews( CView* pSender, LPARAM lHint = 0L,
CObject* pHint = NULL );

        bool      CloseConnectionWindow(CString strLabel);
        void      CloseFeedbackListView( );
        bool      IsConnectionWindowOpen(CString t_strConnectionLabel);
        bool      LoadLayout( );
        bool      OpenConnectionWindow(CString strLabel, bool
bNewLayout = true);
        bool      OpenConnectionsWindow();
        bool      OpenFeedbackWindow( );
        bool      OpenInstantMessengerWindow(CString
strConnectionLabel) ;
        bool      OpenLocalVideoWindow(CString strConnectionLabel) ;
        bool      OpenRemoteVideoWindow(CString strConnectionLabel) ;
        bool      OpenPlaybackWindow( );
        bool      SetFeedbackListView( CMainDoc * MainDoc );
        void      SplashScreen();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CProjectNalayApp)
public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual void Serialize(CArchive& ar);
//}}AFX_VIRTUAL

// Implementation
//{{AFX_MSG(CProjectNalayApp)
afx_msg void OnAppAbout();
afx_msg void OnViewConnections();
afx_msg void OnUpdateViewConnections(CCmdUI* pCmdUI);
afx_msg void OnViewFeedback();
afx_msg void OnUpdateViewFeedback(CCmdUI* pCmdUI);
afx_msg void OnViewVideoPlayback();
afx_msg void OnUpdateViewVideoPlayback(CCmdUI* pCmdUI);
afx_msg void OnViewConfiguration();
afx_msg void OnFileRegisterWithYellowPages();
afx_msg void OnFileUnregisterWithYellowPages();
afx_msg void OnHelpEmailSupport();
afx_msg void OnHelpCheckForUpdates();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
public:
};

////////////////////////////////////
////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
!defined(AFX_PROJECTNALAY_H__BDBDB4F_4B8B_423F_A4F1_7EBC917ADAAC__INCL
UDED_)

#if
!defined(AFX_PROPPAGEAUDIO_H__19B040EB_4FA3_4A11_8D09_4FD96AF3D97B__INC
LUDED_)
#define
AFX_PROPPAGEAUDIO_H__19B040EB_4FA3_4A11_8D09_4FD96AF3D97B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageAudio.h : header file
//

////////////////////////////////////
/////
// CPropPageAudio dialog

class CPropPageAudio : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageAudio)

// Construction
public:
    CPropPageAudio();
    ~CPropPageAudio();

// Dialog Data
    //{AFX_DATA(CPropPageAudio)
    enum { IDD = IDD_PROPPAGE_AUDIO };
    CString        m_DefaultDirectory;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPageAudio)
    public:
        virtual void OnOK();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageAudio)
    virtual BOOL OnInitDialog();
    afx_msg void OnBrowse();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PROPPAGEAUDIO_H__19B040EB_4FA3_4A11_8D09_4FD96AF3D97B__INC
LUDED_)
#if
!defined(AFX_PROPPAGECONNECTIONS_H__10FF4EB2_54D3_46BB_895B_C040C6ACB94
B__INCLUDED_)
#define
AFX_PROPPAGECONNECTIONS_H__10FF4EB2_54D3_46BB_895B_C040C6ACB94B__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageConnections.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageConnections dialog

class CPropPageConnections : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageConnections)

// Construction
public:
    CPropPageConnections();
    ~CPropPageConnections();

// Dialog Data
    //{{AFX_DATA(CPropPageConnections)
    enum { IDD = IDD_PROPPAGE_CONNECTIONS };
    CComboBox     m_OnDoubleClickCtrl;
    int            m_OnDoubleClick;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPropPageConnections)
    public:
        virtual void OnOK();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CPropPageConnections)
    virtual BOOL OnInitDialog();

```

```

    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGECONNECTIONS_H__10FF4EB2_54D3_46BB_895B_C040C6ACB94
B__INCLUDED_
#if
#ifndef AFX_PROPPAGEEMAIL_H__662D96B4_9CD4_427E_85D2_696D0628031A__INC
LUDED_
#define
AFX_PROPPAGEEMAIL_H__662D96B4_9CD4_427E_85D2_696D0628031A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageEmail.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageEmail dialog

class CPropPageEmail : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageEmail)

// Construction
public:
    CPropPageEmail();
    ~CPropPageEmail();

// Dialog Data
    //{{AFX_DATA(CPropPageEmail)
    enum { IDD = IDD_PROPPAGE_EMAIL };
    // NOTE - ClassWizard will add data members here.
    // DO NOT EDIT what you see in these blocks of generated
code !
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPropPageEmail)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions

```

```

//{{AFX_MSG(CPropPageEmail)
    // NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGEEMAIL_H__662D96B4_9CD4_427E_85D2_696D0628031A__INC
LUDED_
#endif
#ifndef AFX_PROPPAGEFEEDBACK_H__A5E25694_CB5E_4C27_BF01_1681FB6CACDE__
INCLUDED_
#define
AFX_PROPPAGEFEEDBACK_H__A5E25694_CB5E_4C27_BF01_1681FB6CACDE__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageFeedback.h : header file
//

////////////////////////////////////
/////
// CPropPageFeedback dialog

class CPropPageFeedback : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageFeedback)

// Construction
public:
    CPropPageFeedback();
    ~CPropPageFeedback();

// Dialog Data
    //{{AFX_DATA(CPropPageFeedback)
    enum { IDD = IDD_PROPPAGE_FEEDBACK };
    BOOL m_ShowDebugFeedback;
    BOOL m_ShowErrorFeedback;
    BOOL m_ShowStatusFeedback;
    BOOL m_ShowWarningFeedback;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPropPageFeedback)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

```

```

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageFeedback)
    // NOTE: the ClassWizard will add member functions here
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGEFEEDBACK_H__A5E25694_CB5E_4C27_BF01_1681FB6CACDE__
    INCLUDED_
#endif
#ifndef AFX_PROPPAGEPHONEMODEM_H__3E83C240_D6F2_4E93_A733_1D1D11290FC9__
    INCLUDED_
#endif
#define AFX_PROPPAGEPHONEMODEM_H__3E83C240_D6F2_4E93_A733_1D1D11290FC9__INCLUDE
D_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPagePhoneModem.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPagePhoneModem dialog

class CPropPagePhoneModem : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPagePhoneModem)

// Construction
public:
    CPropPagePhoneModem();
    ~CPropPagePhoneModem();

    CString m_TAPIDevice;

// Dialog Data
    //{AFX_DATA(CPropPagePhoneModem)
    enum { IDD = IDD_PROPPAGE_PHONEMODEM };
    CComboBox m_DeviceCtrl;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPagePhoneModem)
    public:
        virtual void OnOK();

```

```

        protected:
            virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPropPagePhoneModem)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGEPHONEMODEM_H__3E83C240_D6F2_4E93_A733_1D1D11290FC9__INCLUDED_
#define AFX_PROPPAGERECORDVIDEO_H__F9C8A256_E6E0_48D5_AFOE_CFAEB765B261__INCLUDED_
#define AFX_PROPPAGERECORDVIDEO_H__F9C8A256_E6E0_48D5_AFOE_CFAEB765B261__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageRecordVideo.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageRecordVideo dialog

class CPropPageRecordVideo : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageRecordVideo)

// Construction
public:
    CPropPageRecordVideo();
    ~CPropPageRecordVideo();

// Dialog Data
    //{{AFX_DATA(CPropPageRecordVideo)
    enum { IDD = IDD_PROPPAGE_VIDEORECORD };
    CDateTimeCtrl      m_DefaultRecordDurationCtrl;
    CTime m_DefaultRecordDuration;
    CString            m_DefaultRecordFile;
    int                m_MaxContinuousFiles;
    //}}AFX_DATA

```

```

// Ov rrides
// ClassWizard generate virtual function overrides
//{{AFX_VIRTUAL(CPropPageRecordVideo)
public:
    virtual void OnOK();
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageRecordVideo)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGERECORDVIDEO_H__F9C8A256_E6E0_48D5_AF0E_CFAEB765B26
1_INCLUDED_
#endif
#ifndef AFX_PROPPAGESECURITYPROMPTS_H__C0E7DAD6_F3DA_426F_871F_15D5C9B
90C02_INCLUDED_
#define
AFX_PROPPAGESECURITYPROMPTS_H__C0E7DAD6_F3DA_426F_871F_15D5C9B90C02__IN
CLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageSecurityPrompts.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageSecurityPrompts dialog

class CPropPageSecurityPrompts : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageSecurityPrompts)

// Construction
public:
    CPropPageSecurityPrompts();
    ~CPropPageSecurityPrompts();

    void UpdateCheckBoxes();

// Dialog Data

```

```

//{{AFX_DATA(CPropPageSecurityPrompts)
enum { IDD = IDD_PROPPAGE_SECURITYPROMPTS };
CEdit m_Password2Ctrl;
CEdit m_PasswordCtrl;
CButton m_LaunchCtrl;
CButton m_ExitCtrl;
CButton m_ConnectionCtrl;
CButton m_ConfigurationCtrl;
CButton m_AlarmCtrl;
BOOL m_PwdPromptOnAlarm;
BOOL m_PwdPromptOnConfiguration;
BOOL m_PwdPromptOnConnection;
BOOL m_PwdPromptOnExit;
BOOL m_PwdPromptOnLaunch;
CString m_AppPassword;
CString m_AppPassword2;
BOOL m_PromptOnExit;
BOOL m_EnableAppSecurityPrompt;
//}}AFX_DATA

// Overrides
// ClassWizard generate virtual function overrides
//{{AFX_VIRTUAL(CPropPageSecurityPrompts)
public:
virtual void OnOK();
virtual BOOL OnApply();
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
//}}AFX_VIRTUAL

// Implementation
protected:
// Generated message map functions

//{{AFX_MSG(CPropPageSecurityPrompts)
virtual BOOL OnInitDialog();
afx_msg void OnEnableapplicationsecurityprompt();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PROPPAGESECURITYPROMPTS_H__C0E7DAD6_F3DA_426F_871F_15D5C9B
90C02__INCLUDED_)
#if
!defined(AFX_PROPPAGEUPDATE_H__6D7B8B22_C2D8_46D5_8DCB_023C80ED42FA__IN
CLUDED_)
#define
AFX_PROPPAGEUPDATE_H__6D7B8B22_C2D8_46D5_8DCB_023C80ED42FA__INCLUDED_

```

```

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageUpdate.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageUpdate dialog

class CPropPageUpdate : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageUpdate)

// Construction
public:
    CPropPageUpdate();
    ~CPropPageUpdate();

// Dialog Data
    //{AFX_DATA(CPropPageUpdate)
    enum { IDD = IDD_PROPPAGE_UPDATE };
    BOOL m_CheckForUpdateAtLaunch;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPageUpdate)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageUpdate)
        // NOTE: the ClassWizard will add member functions here
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROPPAGEUPDATE_H__6D7B8B22_C2D8_46D5_8DCB_023C80ED42FA__IN
CLUDED_
#define AFX_PROPPAGEUPDATE_H__6D7B8B22_C2D8_46D5_8DCB_023C80ED42FA__IN
CLUDED_
#endif
#ifndef AFX_PROPPAGEVIDEO_H__315B007A_1FEC_4D58_AA37_224A62F3C781__INC
LUDED_
#define AFX_PROPPAGEVIDEO_H__315B007A_1FEC_4D58_AA37_224A62F3C781__INC
LUDED_

```

```

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageVideo.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageVideo dialog

class CPropPageVideo : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageVideo)

// Construction
public:
    CPropPageVideo();
    ~CPropPageVideo();

// Dialog Data
    //{AFX_DATA(CPropPageVideo)
    enum { IDD = IDD_PROPPAGE_VIDEO };
    int         m_Format;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPageVideo)
    public:
        virtual void OnOK();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageVideo)
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_PROPPAGEVIDEO_H__315B007A_1FEC_4D58_AA37_224A62F3C781__INC
LUDED_)
#if
!defined(AFX_PROPPAGEVIDEOPLAYBACK_H__6027CE98_A86D_4082_A46F_B9FD2E313
BD1__INCLUDED_)

```

```

#define
AFX_PROPPAGEVIDEOPLAYBACK_H__6027CE98_A86D_4082_A46F_B9FD2E313BD1__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageVideoPlayback.h : header file
//

/////////////////////////////////////////////////////////////////
/////
// CPropPageVideoPlayback dialog

class CPropPageVideoPlayback : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageVideoPlayback)

// Construction
public:
    CPropPageVideoPlayback();
    ~CPropPageVideoPlayback();

// Dialog Data
    //{AFX_DATA(CPropPageVideoPlayback)
    enum { IDD = IDD_PROPPAGE_VIDEOPLAYBACK };
    int         m_FastForwardSpeed;
    int         m_ForwardSpeed;
    CString     m_DefaultDirectory;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPageVideoPlayback)
    public:
        virtual void OnOK();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageVideoPlayback)
    virtual BOOL OnInitDialog();
    afx_msg void OnBrowse();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
!defined(AFX_PROPPAGEVIDEOPLAYBACK_H__6027CE98_A86D_4082_A46F_B9FD2E313
BD1__INCLUDED_)
#if
!defined(AFX_PROPPAGEYELLOWPAGES_H__9789C3F1_4137_4953_AB67_9E48592DC3B
0__INCLUDED_)
#define
AFX_PROPPAGEYELLOWPAGES_H__9789C3F1_4137_4953_AB67_9E48592DC3B0__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropPageYellowPages.h : header file
//

////////////////////////////////////
/////
// CPropPageYellowPages dialog

class CPropPageYellowPages : public CPropertyPage
{
    DECLARE_DYNCREATE(CPropPageYellowPages)

// Construction
public:
    CPropPageYellowPages();
    ~CPropPageYellowPages();

// Dialog Data
    //{AFX_DATA(CPropPageYellowPages)
    enum { IDD = IDD_PROPPAGE_YELLOWPAGES };
    BOOL m_AutoRegisterYellowPages;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPropPageYellowPages)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CPropPageYellowPages)
    // NOTE: the ClassWizard will add member functions here
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

```

#endif //
!defined(AFX_PROPPAGEYELLOWPAGES_H__9789C3F1_4137_4953_AB67_9E48592DC3B
0_INCLUDED_)
#if
!defined(AFX_REMOTEVIDEOEVENTSVIEW_H__05B61FB2_DE3D_491F_ACFD_ACBFD0357
72C_INCLUDED_)
#define
AFX_REMOTEVIDEOEVENTSVIEW_H__05B61FB2_DE3D_491F_ACFD_ACBFD035772C__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// RemoteVideoEventsView.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CRemoteVideoEventsView view

class CRemoteVideoEventsView : public CListView
{
protected:
    CRemoteVideoEventsView();          // protected constructor used
by dynamic creation
    DECLARE_DYNCREATE(CRemoteVideoEventsView)

// Attributes
public:
    CImageList          m_LargeImageList;
    CImageList          m_SmallImageList;
    CImageList          m_StateImageList;
    CEventsArray        m_Events;

// Operations
public:

    void                OnEditEvent() ;
    bool                GetConnectionLabel(CString& strConnectionLabel);
    int                 GetSelectedEventItem();
    bool                GetSelectedEventLabel(CString& strLabel) ;
    DWORD               GetViewType();
    bool                ReceiveEventsArray(CConnectionMsg* pConnectionMsg);
    bool                SendEventsArray();
    BOOL                SetColumnWidth(int Column, int Width);
    BOOL                SetViewType(DWORD dwViewType);
    bool                UpdateListCtrl();

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CRemoteVideoEventsView)
    public:
        virtual void OnInitialUpdate();
    protected:
        virtual void OnDraw(CDC* pDC);    // overridden to draw this
view

```

```

        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CRemoteVideoEventsView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
protected:
        //{{AFX_MSG(CRemoteVideoEventsView)
        afx_msg void OnConnectionConnect();
        afx_msg void OnConnectionDisconnect();
        afx_msg void OnConnectionDeleteEvent();
        afx_msg void OnConnectionNewAlarm();
        afx_msg void OnConnectionNewEvent();
        afx_msg void OnViewLargeicons();
        afx_msg void OnViewSmallicons();
        afx_msg void OnViewList();
        afx_msg void OnViewDetails();
        afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnUpdateConnectionConnect(CCmdUI* pCmdUI);
        afx_msg void OnUpdateConnectionDisconnect(CCmdUI* pCmdUI);
        afx_msg void OnDestroy();
        afx_msg void OnUpdateConnectionNewAlarm(CCmdUI* pCmdUI);
        afx_msg void OnUpdateConnectionNewEvent(CCmdUI* pCmdUI);
        afx_msg void OnUpdateConnectionDeleteEvent(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_REMOTEVIDEOEVENTSVIEW_H__05B61FB2_DE3D_491F_ACFD_ACBFD0357
72C__INCLUDED_
#define AFX_REMOTEVIDEOEVENTSVIEW_H__05B61FB2_DE3D_491F_ACFD_ACBFD0357
72C__INCLUDED_
#endif
#ifndef AFX_REMOTEVIDEOFRAME_H__E9D7F216_7770_488C_B571_C6AD758D8C5B__
INCLUDED_
#define AFX_REMOTEVIDEOFRAME_H__E9D7F216_7770_488C_B571_C6AD758D8C5B__
INCLUDED_
#endif
#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// RemoteVideoFrame.h : header file
//

```

```

////////////////////////////////////
/////
// CRemoteVideoFrame frame

class CRemoteVideoFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CRemoteVideoFrame)
protected:
    CRemoteVideoFrame();           // protected constructor used by
dynamic creation

// Attributes
public:
    CSplitterWnd m_wndSplitter;
    CToolBar      m_wndToolBar;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CRemoteVideoFrame)
protected:
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //{AFX_VIRTUAL

// Implementation
protected:
    virtual ~CRemoteVideoFrame();

    // Generated message map functions
    //{AFX_MSG(CRemoteVideoFrame)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //{AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
/////

//{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
#ifndef AFX_REMOTEVIDEOFRAME_H__E9D7F216_7770_488C_B571_C6AD758D8C5B__
INCLUDED_

#include "stdafx.h"
#include "Project Nayal.h"

#include "YellowPages.h"

```

```

// #import "..\Yellow Pages\VPD.ocx" named_guids no_namespace

// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
// regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

#include "stdafx.h"
#include "activemovie3.h"

////////////////////////////////////
/////
// CActiveMovie3

IMPLEMENT_DYNCREATE(CActiveMovie3, CWnd)

////////////////////////////////////
/////
// CActiveMovie3 properties

////////////////////////////////////
/////
// CActiveMovie3 operations

void CActiveMovie3::AboutBox()
{
    InvokeHelper(0xfffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CActiveMovie3::Run()
{
    InvokeHelper(0x60020001, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CActiveMovie3::Pause()
{
    InvokeHelper(0x60020002, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CActiveMovie3::Stop()
{
    InvokeHelper(0x60020003, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CActiveMovie3::GetImageSourceWidth()
{
    long result;
    InvokeHelper(0x4, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

```

```
long CActiveMovie3::GetImageSourceHeight()
{
    long result;
    InvokeHelper(0x5, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetAuthor()
{
    CString result;
    InvokeHelper(0x6, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetTitle()
{
    CString result;
    InvokeHelper(0x7, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetCopyright()
{
    CString result;
    InvokeHelper(0x8, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetDescription()
{
    CString result;
    InvokeHelper(0x9, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetRating()
{
    CString result;
    InvokeHelper(0xa, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

CString CActiveMovie3::GetFileName()
{
    CString result;
    InvokeHelper(0xb, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}
```

```

void CActiveMovie3::SetFileName(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0xb, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

double CActiveMovie3::GetDuration()
{
    double result;
    InvokeHelper(0xc, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
        NULL);
    return result;
}

double CActiveMovie3::GetCurrentPosition()
{
    double result;
    InvokeHelper(0xd, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetCurrentPosition(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0xd, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

long CActiveMovie3::GetPlayCount()
{
    long result;
    InvokeHelper(0xe, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetPlayCount(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0xe, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

double CActiveMovie3::GetSelectionStart()
{
    double result;
    InvokeHelper(0xf, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetSelectionStart(double newValue)

```

```

{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0xf, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

double CActiveMovie3::GetSelectionEnd()
{
    double result;
    InvokeHelper(0x10, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetSelectionEnd(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x10, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

long CActiveMovie3::GetCurrentState()
{
    long result;
    InvokeHelper(0x11, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

double CActiveMovie3::GetRate()
{
    double result;
    InvokeHelper(0x12, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetRate(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x12, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

long CActiveMovie3::GetVolume()
{
    long result;
    InvokeHelper(0x13, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetVolume(long nNewValue)
{

```

```

        static BYTE parms[] =
            VTS_I4;
        InvokeHelper(0x13, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            nNewValue);
    }

long CActiveMovie3::GetBalance()
{
    long result;
    InvokeHelper(0x14, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetBalance(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x14, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CActiveMovie3::GetEnableContextMenu()
{
    BOOL result;
    InvokeHelper(0x15, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetEnableContextMenu(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x15, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetShowDisplay()
{
    BOOL result;
    InvokeHelper(0x16, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetShowDisplay(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x16, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetShowControls()
{
    BOOL result;

```

```

        InvokeHelper(0x17, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
NULL);
        return result;
    }

void CActiveMovie3::SetShowControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x17, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetShowPositionControls()
{
    BOOL result;
    InvokeHelper(0x18, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
NULL);
    return result;
}

void CActiveMovie3::SetShowPositionControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x18, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetShowSelectionControls()
{
    BOOL result;
    InvokeHelper(0x19, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
NULL);
    return result;
}

void CActiveMovie3::SetShowSelectionControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x19, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetShowTracker()
{
    BOOL result;
    InvokeHelper(0x1a, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
NULL);
    return result;
}

void CActiveMovie3::SetShowTracker(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;

```

```

        InvokeHelper(0x1a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                     bNewValue);
    }

    BOOL CActiveMovie3::GetEnablePositionControls()
    {
        BOOL result;
        InvokeHelper(0x1b, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
                     NULL);
        return result;
    }

    void CActiveMovie3::SetEnablePositionControls(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x1b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                     bNewValue);
    }

    BOOL CActiveMovie3::GetEnableSelectionControls()
    {
        BOOL result;
        InvokeHelper(0x1c, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
                     NULL);
        return result;
    }

    void CActiveMovie3::SetEnableSelectionControls(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x1c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                     bNewValue);
    }

    BOOL CActiveMovie3::GetEnableTracker()
    {
        BOOL result;
        InvokeHelper(0x1d, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
                     NULL);
        return result;
    }

    void CActiveMovie3::SetEnableTracker(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x1d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                     bNewValue);
    }

    BOOL CActiveMovie3::GetAllowHideDisplay()
    {
        BOOL result;
        InvokeHelper(0x1e, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
                     NULL);
    }

```

```

        return result;
    }

void CActiveMovie3::SetAllowHideDisplay(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x1e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetAllowHideControls()
{
    BOOL result;
    InvokeHelper(0x1f, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetAllowHideControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x1f, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CActiveMovie3::GetDisplayMode()
{
    long result;
    InvokeHelper(0x20, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetDisplayMode(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x20, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CActiveMovie3::GetAllowChangeDisplayMode()
{
    BOOL result;
    InvokeHelper(0x21, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetAllowChangeDisplayMode(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x21, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

```

```

}

LPUNKNOWN CActiv Movie3::GetFilterGraph()
{
    LPUNKNOWN result;
    InvokeHelper(0x22, DISPATCH_PROPERTYGET, VT_UNKNOWN,
(void*)&result, NULL);
    return result;
}

void CActiveMovie3::SetFilterGraph(LPUNKNOWN newValue)
{
    static BYTE parms[] =
        VTS_UNKNOWN;
    InvokeHelper(0x22, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

LPDISPATCH CActiveMovie3::GetFilterGraphDispatch()
{
    LPDISPATCH result;
    InvokeHelper(0x23, DISPATCH_PROPERTYGET, VT_DISPATCH,
(void*)&result, NULL);
    return result;
}

unsigned long CActiveMovie3::GetDisplayForeColor()
{
    unsigned long result;
    InvokeHelper(0x24, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CActiveMovie3::SetDisplayForeColor(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x24, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

unsigned long CActiveMovie3::GetDisplayBackColor()
{
    unsigned long result;
    InvokeHelper(0x25, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CActiveMovie3::SetDisplayBackColor(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x25, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

```

```
long CActiveMovie3::GetMovieWindowSize()
{
    long result;
    InvokeHelper(0x26, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CActiveMovie3::SetMovieWindowSize(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x26, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CActiveMovie3::GetFullScreenMode()
{
    BOOL result;
    InvokeHelper(0x27, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}

void CActiveMovie3::SetFullScreenMode(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x27, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetAutoStart()
{
    BOOL result;
    InvokeHelper(0x28, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}

void CActiveMovie3::SetAutoStart(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x28, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CActiveMovie3::GetAutoRewind()
{
    BOOL result;
    InvokeHelper(0x29, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}
```

```

void CActiveMovie3::SetAutoRewind(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x29, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CActiveMovie3::GetHWnd()
{
    long result;
    InvokeHelper(DISPID_HWND, DISPATCH_PROPERTYGET, VT_I4,
        (void*)&result, NULL);
    return result;
}

long CActiveMovie3::GetAppearance()
{
    long result;
    InvokeHelper(DISPID_APPEARANCE, DISPATCH_PROPERTYGET, VT_I4,
        (void*)&result, NULL);
    return result;
}

void CActiveMovie3::SetAppearance(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(DISPID_APPEARANCE, DISPATCH_PROPERTYPUT, VT_EMPTY,
        NULL, parms,
        nNewValue);
}

long CActiveMovie3::GetBorderStyle()
{
    long result;
    InvokeHelper(0x2a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CActiveMovie3::SetBorderStyle(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x2a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CActiveMovie3::GetEnabled()
{
    BOOL result;
    InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

```

```

void CActiveMovie3::SetEnabled(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYPUT, VT_EMPTY,
        NULL, parms,
            bNewValue);
}

BOOL CActiveMovie3::IsSoundCardEnabled()
{
    BOOL result;
    InvokeHelper(0x35, DISPATCH_METHOD, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

long CActiveMovie3::GetReadyState()
{
    long result;
    InvokeHelper(DISPID_READYSTATE, DISPATCH_PROPERTYGET, VT_I4,
        (void*)&result, NULL);
    return result;
}

LPDISPATCH CActiveMovie3::GetMediaPlayer()
{
    LPDISPATCH result;
    InvokeHelper(0x457, DISPATCH_PROPERTYGET, VT_DISPATCH,
        (void*)&result, NULL);
    return result;
}
// AdvConnectionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "AdvConnectionDlg.h"

#include "SimpleVideo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CActiveConnectionDlg dialog

CAdvConnectionDlg::CAdvConnectionDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAdvConnectionDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAdvConnectionDlg)
    m_VideoPort = 0;

```

```

        //}}AFX_DATA_INIT
    }

void CAdvConnectionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAdvConnectionDlg)
    DDX_Control(pDX, IDC_VIDEOPROFILE, m_VideoProfileCtrl);
    DDX_Text(pDX, IDC_VIDEOPORT, m_VideoPort);
    DDV_MinMaxInt(pDX, m_VideoPort, 1, 20000);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAdvConnectionDlg, CDialog)
    //{{AFX_MSG_MAP(CAdvConnectionDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CAdvConnectionDlg message handlers

BOOL CAdvConnectionDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    CSimpleVideo t_SimpleVideo;
    CStringArray t_arrProfiles;

    t_SimpleVideo.GetProfiles(t_arrProfiles);
/*
    for ( int i = 0; i < t_arrProfiles.GetSize(); i++ )
        m_VideoProfileCtrl.AddString(t_arrProfiles.ElementAt(i));

    m_VideoProfileCtrl.SetCurSel(m_VideoProfile );
*/

    int i, j;

    for ( j = 0, i = 0; i < t_arrProfiles.GetSize(); i++ )
    {
        // only use profiles 0, 8 & 11 for slow, medium and fast
connections
        if ( i == 0 || i == 8 || i == 11 )
        {
            m_VideoProfileCtrl.AddString(t_arrProfiles.ElementAt(i));
            m_VideoProfileCtrl.SetItemData(j, (DWORD) i);
            j++;
        }
    }

    for ( i = 0; i < m_VideoProfileCtrl.GetCount(); i++ )
    {
        int t_VideoProfile;

```

```

        t_VideoProfile = (int) m_VideoProfileCtrl.GetItemData(i);
        if ( t_VideoProfile == m_VideoProfile )
            m_VideoProfileCtrl.SetCurSel(i);
    }
/*
    switch ( m_VideoProfile )
    {
    case 0:
        m_VideoProfileCtrl.SetCurSel(0);
        break;
    case 8:
        m_VideoProfileCtrl.SetCurSel(1);
        break;
    case 11:
        m_VideoProfileCtrl.SetCurSel(2);
        break;
    }
*/

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
    }

```

```

void CAdvConnectionDlg::OnOK()
{
    int t_CurSel;

    t_CurSel = m_VideoProfileCtrl.GetCurSel();
    m_VideoProfile = m_VideoProfileCtrl.GetItemData(t_CurSel );

    // m_VideoProfile = m_VideoProfileCtrl.GetCurSel();
    /*
        int t_nIndex;

        t_nIndex = m_VideoProfileCtrl.GetCurSel();

        switch ( t_nIndex )
        {
        case 0:
            m_VideoProfile = 0;
            break;
        case 1:
            m_VideoProfile = 8;
            break;
        case 2:
            m_VideoProfile = 11;
            break;
        }
    */
    CDialog::OnOK();
}

```

```
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++
```

```
// NOTE: Do not modify the contents of this file.  If this class is
regenerated by
// Microsoft Visual C++, your modifications will be overwritten.
```

```
#include "stdafx.h"
#include "amtapi.h"
```

```
////////////////////////////////////
/////
// CamTapi
```

```
IMPLEMENT_DYNCREATE(CamTapi, CWnd)
```

```
////////////////////////////////////
/////
// CamTapi properties
```

```
////////////////////////////////////
/////
// CamTapi operations
```

```
long CamTapi::GetNumberOfLines()
{
    long result;
    InvokeHelper(0x1, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}
```

```
long CamTapi::GetDevCaps()
{
    long result;
    InvokeHelper(0x2, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}
```

```
CString CamTapi::GetLineName(long LineId)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x3, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
    LineId);
    return result;
}
```

```
void CamTapi::TapiReset()
{
    InvokeHelper(0x4, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}
```

```

CString CamTapi::GetLineName()
{
    CString result;
    InvokeH_lper(0x5, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

void CamTapi::SetLineName(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x5, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

void CamTapi::About()
{
    InvokeHelper(0xfffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CamTapi::HangUp()
{
    InvokeHelper(0x7, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CamTapi::MakeCall(LPCTSTR dialNumber)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x8, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        dialNumber);
}

CString CamTapi::GetCallState()
{
    CString result;
    InvokeHelper(0x9, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

void CamTapi::Answer()
{
    InvokeHelper(0xa, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CamTapi::ShowLocationDialog(long hWnd, LPCTSTR Number)
{
    static BYTE parms[] =
        VTS_I4 VTS_BSTR;
    InvokeHelper(0xb, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        hWnd, Number);
}

CString CamTapi::TranslateNumber(BSTR* Number)

```

```

{
    CString result;
    static BYTE parms[] =
        VTS_PBSTR;
    InvokeHelper(0xc, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        Number);
    return result;
}

long CamTapi::GetMediaMode()
{
    long result;
    InvokeHelper(0xd, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CamTapi::SetMediaMode(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0xd, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

long CamTapi::GetCallPrivilege()
{
    long result;
    InvokeHelper(0xe, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CamTapi::SetCallPrivilege(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0xe, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

long CamTapi::GetCommHandle()
{
    long result;
    InvokeHelper(0xf, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CamTapi::GetLineWaveInID()
{
    long result;
    InvokeHelper(0x10, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

```

```

long CamTapi::GetLineWaveOutID()
{
    long result;
    InvokeHelper(0x11, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CamTapi::GenerateDigits(LPCTSTR Digits, long msDuration)
{
    static BYTE parms[] =
        VTS_BSTR VTS_I4;
    InvokeHelper(0x12, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        Digits, msDuration);
}

void CamTapi::Dial(LPCTSTR dialNumber)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x13, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        dialNumber);
}

CString CamTapi::ShowModemDialog(long hWnd, LPCTSTR DisplaySettings)
{
    CString result;
    static BYTE parms[] =
        VTS_I4 VTS_BSTR;
    InvokeHelper(0x14, DISPATCH_METHOD, VT_BSTR, (void*)&result,
    parms,
        hWnd, DisplaySettings);
    return result;
}

CString CamTapi::GetCurrentSettings()
{
    CString result;
    InvokeHelper(0x15, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
    NULL);
    return result;
}

void CamTapi::SetCurrentSettings(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x15, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

BOOL CamTapi::GetLineOpen()
{
    BOOL result;
    InvokeHelper(0x16, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
}

```

```

        return result;
    }

void CamTapi::SetLineOpen(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x16, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CamTapi::GetTapiNegotiatedVersion()
{
    long result;
    InvokeHelper(0x17, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CamTapi::ShowLineDialog(long hWnd)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x18, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        hWnd);
}
// Machine generated IDispatch wrapper class(es) created by Microsoft
// Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
// regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

#include "stdafx.h"
#include "amwave.h"

////////////////////////////////////
/////
// CamWave

IMPLEMENT_DYNCREATE(CamWave, CWnd)

////////////////////////////////////
/////
// CamWave properties

////////////////////////////////////
/////
// CamWave operations

CString CamWave::GetPlayFilename()
{
    CString result;
    InvokeHelper(0x1, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
        NULL);
    return result;
}

```

```

}

void CamWave::SetPlayFilename(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x1, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

short CamWave::GetPlayVolume()
{
    short result;
    InvokeHelper(0x2, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
        NULL);
    return result;
}

void CamWave::SetPlayVolume(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x2, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

CString CamWave::GetRecordFilename()
{
    CString result;
    InvokeHelper(0x3, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result,
        NULL);
    return result;
}

void CamWave::SetRecordFilename(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x3, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

short CamWave::GetRecordLevel()
{
    short result;
    InvokeHelper(0x4, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
        NULL);
    return result;
}

void CamWave::SetRecordLevel(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x4, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

```

```

short CamWave::GetSilenceLevel()
{
    short result;
    InvokeHelper(0x5, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
    NULL);
    return result;
}

void CamWave::SetSilenceLevel(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x5, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

short CamWave::GetSilenceTimer()
{
    short result;
    InvokeHelper(0x6, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
    NULL);
    return result;
}

void CamWave::SetSilenceTimer(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x6, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

void CamWave::Play(long WaveOutID)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        WaveOutID);
}

void CamWave::Record(long WaveInID)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x8, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        WaveInID);
}

void CamWave::StopPlay()
{
    InvokeHelper(0x9, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CamWave::StopRecord()
{

```

```

        InvokeHelper(0xa, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
    }

long CamWave::GetRecordFormat()
{
    long result;
    InvokeHelper(0xb, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CamWave::SetRecordFormat(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0xb, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

short CamWave::GetWaveInNumDevs()
{
    short result;
    InvokeHelper(0xc, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
    NULL);
    return result;
}

short CamWave::GetWaveOutNumDevs()
{
    short result;

    InvokeHelper(0xd, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
    NULL);
    return result;
}

CString CamWave::WaveInGetName(short DeviceID)
{
    CString result;
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0xe, DISPATCH_METHOD, VT_BSTR, (void*)&result,
    parms,
        DeviceID);
    return result;
}

CString CamWave::WaveOutGetName(short DeviceID)
{
    CString result;
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0xf, DISPATCH_METHOD, VT_BSTR, (void*)&result,
    parms,
        DeviceID);
    return result;
}

```

```

void CamWave::About()
{
    InvokeHelp r(0xfffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CamWave::WaveInGetCaps(short DeviceID)
{
    long result;
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x10, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
        DeviceID);
    return result;
}

long CamWave::WaveOutGetCaps(short DeviceID)
{
    long result;
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x11, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
        DeviceID);
    return result;
}

short CamWave::GetPlayBufferLength()
{
    short result;
    InvokeHelper(0x12, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
        NULL);
    return result;
}

void CamWave::SetPlayBufferLength(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x12, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

short CamWave::GetRecordBufferLength()
{
    short result;
    InvokeHelper(0x13, DISPATCH_PROPERTYGET, VT_I2, (void*)&result,
        NULL);
    return result;
}

void CamWave::SetRecordBufferLength(short nNewValue)
{
    static BYTE parms[] =
        VTS_I2;
    InvokeHelper(0x13, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

```

```

// AppSecurityDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "AppSecurityDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CAppSecurityDlg dialog

CAppSecurityDlg::CAppSecurityDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAppSecurityDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAppSecurityDlg)
    m_Password = _T("");
    //}}AFX_DATA_INIT
}

void CAppSecurityDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAppSecurityDlg)
    DDX_Text(pDX, IDC_PASSWORD, m_Password);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAppSecurityDlg, CDialog)
    //{{AFX_MSG_MAP(CAppSecurityDlg)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CAppSecurityDlg message handlers
// AudioDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "AudioDlg.h"

#include "SimpleDirectAudio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE

```

```

static char THIS_FILE[] = __FILE__;
#endif

extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CAudioDlg dialog

CAudioDlg::CAudioDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAudioDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CAudioDlg)
    m_AudioFile = _T("");
   //}}AFX_DATA_INIT
}

void CAudioDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAudioDlg)
    DDX_Control(pDX, IDC_RECORD, m_Record);
    DDX_Text(pDX, IDC_AUDIOFILE, m_AudioFile);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAudioDlg, CDialog)
    {{{AFX_MSG_MAP(CAudioDlg)
        ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
        ON_BN_CLICKED(IDC_TEST, OnTest)
        ON_BN_CLICKED(IDC_RECORD, OnRecord)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CAudioDlg message handlers

BOOL CAudioDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    UpdateData(FALSE);
    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CAudioDlg::OnOK()
{

```

```

        // TODO: Add extra validation here
        UpdateData(TRUE);

        CDialog::OnOK();
    }

void CAudioDlg::OnBrowse()
{
    char BASED_CODE t_szFilter[] = "Audio Files (*.wav)|*.wav|All
Files (*.*)|*.*||";
    CString t_strAudioFile;

    t_strAudioFile.Format("%s\\*.wav",
gm_AppConfig.m_DefaultAudioDirectory);

    CFileDialog t_FileDialog(TRUE, ".WAV", t_strAudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);
    // CFileDialog t_FileDialog(TRUE, ".WAV", m_AudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);

    if ( t_FileDialog.DoModal() == IDOK )
    {
        UpdateData(TRUE);
        m_AudioFile = t_FileDialog.m_ofn.lpstrFile;
        UpdateData(FALSE);
    }
}

void CAudioDlg::OnTest()
{
    CSimpleDirectAudio t_SimpleDirectAudio;

    UpdateData(TRUE);

    t_SimpleDirectAudio QuickPlayAudioFile(m_AudioFile);
}

void CAudioDlg::OnRecord()
{
    char BASED_CODE t_szFilter[] = "Audio Files (*.wav)|*.wav|All
Files (*.*)|*.*||";
    CSimpleDirectAudio t_SimpleDirectAudio;
    CString t_strRecordFile;
    CString t_strAudioFile;

    t_strAudioFile.Format("%s\\*.wav",
gm_AppConfig.m_DefaultAudioDirectory);

    CFileDialog t_FileDialog(FALSE, ".WAV", t_strAudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);
    // CFileDialog t_FileDialog(FALSE, ".WAV", m_AudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);

    if ( t_FileDialog.DoModal() == IDOK )
    {
        t_strRecordFile = t_FileDialog.m_ofn.lpstrFile;
    }
}

```

```

        if (
t_SimpleDirectAudio.QuickRecordAudioFile(t_strRecordFile) )
        {
            UpdateData(TRUE);
            m_AudioFile = t_strRecordFile;
            UpdateData(FALSE);
        }
    }
}
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{AFX_MSG_MAP(CChildFrame)
    // NOTE - the ClassWizard will add and remove mapping
macros here.
    //      DO NOT EDIT what you see in these blocks of generated
code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
    // TODO: add member initialization code here
}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

```

```

        if( !CMDIChildWnd::PreCreateWindow(cs) )
            return FALSE;

        return TRUE;
    }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif // _DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CChildFrame message handlers
// ConnectionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "ConnectionDlg.h"
#include "AdvConnectionDlg.h"
#include "MotionDetectionSettingsDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CConnectionsArray gm_Connections;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CConnectionDlg dialog

CConnectionDlg::CConnectionDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CConnectionDlg::IDD, pParent)

```

```

{
    //{AFX_DATA_INIT(CConnectionDlg)
    m_ConnectionMethod = -1;
    m_ConnectionType = -1;
    m_DialUpNumber = _T("");
    m_Password = _T("");
    m_Username = _T("");
    m_YellowPagesEntry = _T("");
    m_Label = _T("");
    m_Password2 = _T("");
    //}}AFX_DATA_INIT

    m_bNewConnection = true;
}

void CConnectionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CConnectionDlg)
    DDX_Control(pDX, IDC_MOTIONDETECTION, m_MotionDetection);
    DDX_Control(pDX, IDC_ADVANCED, m_Advanced);
    DDX_Control(pDX, IDC_KEYDATALABEL, m_KeyDataLabelCtrl);
    DDX_Control(pDX, IDC_AUDIODEVICESLABEL, m_AudioDevicesLabel);
    DDX_Control(pDX, IDC_VIDEODEVICES, m_VideoDevicesCtrl);
    DDX_Control(pDX, IDC_AUDIODEVICES, m_AudioDevicesCtrl);
    DDX_Control(pDX, IDC_CONNECTIONLABEL, m_LabelCtrl);
    DDX_Control(pDX, IDC_IPADDRESS, m_IPAddress);
    DDX_CBIndex(pDX, IDC_CONNECTIONMETHOD, m_ConnectionMethod);
    DDX_CBIndex(pDX, IDC_CONNECTIONTYPE, m_ConnectionType);
    DDX_Text(pDX, IDC_DIALUPNUMBER, m_DialUpNumber);
    DDX_Text(pDX, IDC_PASSWORD, m_Password);
    DDX_Text(pDX, IDC_USERNAME, m_Username);
    DDX_Text(pDX, IDC_YELLOWPAGESENTRY, m_YellowPagesEntry);
    DDX_Text(pDX, IDC_CONNECTIONLABEL, m_Label);
    DDX_Text(pDX, IDC_PASSWORD2, m_Password2);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CConnectionDlg, CDialog)
    //{AFX_MSG_MAP(CConnectionDlg)
    ON_BN_CLICKED(IDC_ADVANCED, OnAdvanced)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    ON_BN_CLICKED(IDC_MOTIONDETECTION, OnMotiondetection)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CConnectionDlg message handlers

BOOL CConnectionDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

```

```

CComboBox* t_ctlCombo;
CString t_str;

t_ctlCombo = (CComboBox* ) GetDlgItem(IDC_CONNECTIONTYPE);
t_str.LoadString(IDS_VIDEOSURVEILLANCE);
t_ctlCombo->AddString(t_str);
t_str.LoadString(IDS_INSTANTMESSENGER);
t_ctlCombo->AddString(t_str);
t_str.LoadString(IDS_VIDEOCONFERENCE);
t_ctlCombo->AddString(t_str);

t_ctlCombo = (CComboBox* ) GetDlgItem(IDC_CONNECTIONMETHOD);
t_str.LoadString(IDS_LOCALTCPIP);
t_ctlCombo->AddString(t_str);
// t_str.LoadString(IDS_LOCALMODEM);
// t_ctlCombo->AddString(t_str);
t_str.LoadString(IDS_IPADDRESS);
t_ctlCombo->AddString(t_str);
t_str.LoadString(IDS_YELLOWPAGES);
t_ctlCombo->AddString(t_str);
// t_str.LoadString(IDS_DIALUP);
// t_ctlCombo->AddString(t_str);

////////////////////////////////////
// Seed Audio and Video Devices
//
CStringArray t_arrDevices;
CSimpleVideo t_SimpleVideo;

t_SimpleVideo.GetVideoCaptureDevices(t_arrDevices);
for (int i = 0; i < t_arrDevices.GetSize(); i++)
    m_VideoDevicesCtrl.AddString(t_arrDevices.ElementAt(i));

t_arrDevices.RemoveAll();

t_SimpleVideo.GetAudioCaptureDevices(t_arrDevices);
for (i = 0; i < t_arrDevices.GetSize(); i++)
    m_AudioDevicesCtrl.AddString(t_arrDevices.ElementAt(i));
//
////////////////////////////////////

UpdateConnectionInformationControls();
UpdateData(FALSE);

// On a new connection set the default video and audio device
// On an existing connection disable the label
if ( m_bNewConnection )
{
    CString t_strDriverName;
    t_strDriverName.LoadString(IDS_DEFAULTVIDEODRIVER);
    m_VideoDevicesCtrl.SelectString(0, "SPCA");
    t_strDriverName.LoadString(IDS_DEFAULTAUDIODRIVER);
    m_AudioDevicesCtrl.SelectString(0, "SPCA");
}
else
{

```

```

        m_LabelCtrl.EnableWindow(FALSE);
    }

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

bool CConnectionDlg::UpdateConnectionInformationControls()
{
    CWnd* t_ctlWnd;
    CString t_str;

    switch ( m_ConnectionInfo->m_ConnectionMethod )
    {
    case CONNECTION_IPADDRESS:
        t_ctlWnd = GetDlgItem(IDC_YELLOWPAGESENTRY);

        t_ctlWnd->ShowWindow(SW_HIDE);
        t_ctlWnd = GetDlgItem(IDC_DIALUPNUMBER);
        t_ctlWnd->ShowWindow(SW_HIDE);
        t_ctlWnd = GetDlgItem(IDC_IPADDRESS);
        t_ctlWnd->ShowWindow(SW_SHOW);
        t_ctlWnd = GetDlgItem(IDC_BROWSE);
        t_ctlWnd->ShowWindow(SW_HIDE);
        t_ctlWnd = GetDlgItem(IDC_KEYDATALABEL);
        t_ctlWnd->ShowWindow(SW_SHOW);
        t_str.LoadString(IDS_IPADDRESS);
        t_ctlWnd->SetWindowText(t_str);
        m_VideoDevicesCtrl.ShowWindow(SW_HIDE);
        m_AudioDevicesCtrl.ShowWindow(SW_HIDE);
        m_AudioDevicesLabel.ShowWindow(SW_HIDE);
        m_MotionDetection.EnableWindow(FALSE);
//        m_Advanced.EnableWindow(FALSE);
        break;

    case CONNECTION_YELLOWPAGES:
        t_ctlWnd = GetDlgItem(IDC_YELLOWPAGESENTRY);
        t_ctlWnd->ShowWindow(SW_SHOW);
        t_ctlWnd = GetDlgItem(IDC_DIALUPNUMBER);
        t_ctlWnd->ShowWindow(SW_HIDE);
        t_ctlWnd = GetDlgItem(IDC_IPADDRESS);
        t_ctlWnd->ShowWindow(SW_HIDE);
        t_ctlWnd = GetDlgItem(IDC_BROWSE);
        t_ctlWnd->ShowWindow(SW_SHOW);
        t_ctlWnd = GetDlgItem(IDC_KEYDATALABEL);
        t_ctlWnd->ShowWindow(SW_SHOW);
        t_str.LoadString(IDS_YELLOWPAGES);
        t_ctlWnd->SetWindowText(t_str);
        m_VideoDevicesCtrl.ShowWindow(SW_HIDE);
        m_AudioDevicesCtrl.ShowWindow(SW_HIDE);
        m_AudioDevicesLabel.ShowWindow(SW_HIDE);
        m_MotionDetection.EnableWindow(FALSE);
//        m_Advanced.EnableWindow(FALSE);
        break;

    case CONNECTION_LOCALTCPIP:
    default:

```

```

t_ctlWnd = GetDlgItem(IDC_YELLOWPAGESEENTRY);
t_ctlWnd->ShowWindow(SW_HIDE);
t_ctlWnd = GetDlgItem(IDC_DIALUPNUMBER);
t_ctlWnd->ShowWindow(SW_HIDE);

t_ctlWnd = GetDlgItem(IDC_IPADDRESS);
t_ctlWnd->ShowWindow(SW_HIDE);
t_ctlWnd = GetDlgItem(IDC_BROWSE);
t_ctlWnd->ShowWindow(SW_HIDE);
t_ctlWnd = GetDlgItem(IDC_KEYDATALABEL);
t_ctlWnd->ShowWindow(SW_SHOW);
t_str.LoadString(IDS_VIDEODEVICE);
t_ctlWnd->SetWindowText(t_str);
m_MotionDetection.EnableWindow(TRUE);
switch (m_ConnectionInfo->m_ConnectionType )
{
case CONTYPE_VIDEOSURVEILLANCE:
    m_KeyDataLabelCtrl.ShowWindow(SW_SHOW);
    m_VideoDevicesCtrl.ShowWindow(SW_SHOW);
    m_AudioDevicesCtrl.ShowWindow(SW_SHOW);
    m_AudioDevicesLabel.ShowWindow(SW_SHOW);
    break;
default:
    m_KeyDataLabelCtrl.ShowWindow(SW_HIDE);
    m_VideoDevicesCtrl.ShowWindow(SW_HIDE);
    m_AudioDevicesCtrl.ShowWindow(SW_HIDE);
    m_AudioDevicesLabel.ShowWindow(SW_HIDE);
    break;
}
// m_Advanced.EnableWindow(TRUE);
break;
}

return true;
}

BOOL CConnectionDlg::UpdateData( BOOL bSaveAndValidate )
{
    BOOL t_bReturn = FALSE;
    int t_nCurSel;

    if ( bSaveAndValidate )
    {
        t_bReturn = CDialog::UpdateData(bSaveAndValidate );
        m_ConnectionInfo->m_Label = m_Label;
        m_IPAddress.GetWindowText(m_ConnectionInfo->m_IPAddress);
        m_ConnectionInfo->m_YellowPagesEntry = m_YellowPagesEntry;
        m_ConnectionInfo->m_DialUpNumber = m_DialUpNumber;
        m_ConnectionInfo->m_ConnectionMethod = m_ConnectionMethod ;
        m_ConnectionInfo->m_ConnectionType = m_ConnectionType;
        m_ConnectionInfo->m_Username = m_Username;
        m_ConnectionInfo->m_Password = m_Password;

        t_nCurSel = m_VideoDevicesCtrl.GetCurSel();
        if ( t_nCurSel != CB_ERR )

```

```

        m_VideoDevicesCtrl.GetLBText(t_nCurSel ,
m_ConnectionInfo->m_SimpleVideo.m_strVideoDevice);

        t_nCurSel = m_AudioDevicesCtrl.GetCurSel();
        if ( t_nCurSel != CB_ERR )
            m_AudioDevicesCtrl.GetLBText(t_nCurSel ,
m_ConnectionInfo->m_SimpleVideo.m_strAudioDevice);
    }
    else
    {
        m_Label = m_ConnectionInfo->m_Label ;
        m_IPAddress.SetWindowText(m_ConnectionInfo->m_IPAddress );
        m_YellowPagesEntry = m_ConnectionInfo->m_YellowPagesEntry ;
        m_DialUpNumber = m_ConnectionInfo->m_DialUpNumber ;
        m_ConnectionMethod = m_ConnectionInfo->m_ConnectionMethod
;

        m_ConnectionType = m_ConnectionInfo->m_ConnectionType ;
        m_Username = m_ConnectionInfo->m_Username ;
        m_Password = m_ConnectionInfo->m_Password ;

        m_VideoDevicesCtrl.SelectString(-1, m_ConnectionInfo-
>m_SimpleVideo.m_strVideoDevice);
        m_AudioDevicesCtrl.SelectString(-1, m_ConnectionInfo-
>m_SimpleVideo.m_strAudioDevice);

        t_bReturn = CDialog::UpdateData(bSaveAndValidate );
    }

    return t_bReturn ;
}

void CConnectionDlg::OnOK()
{
    CString strMessage;

    UpdateData();

    // Make sure that a video and audio device is selected
    // if the Connection Method is Local
    if ( m_ConnectionMethod == CONNECTION_LOCALTCPIP )
    {
        int t_nCurSel;
        t_nCurSel = m_VideoDevicesCtrl.GetCurSel();
        if ( t_nCurSel == CB_ERR )
        {
            strMessage.LoadString(IDS_ERR_MUSTSELECTVIDEODEVICE);
            MessageBox(strMessage);
            return;
        }

        // Also need audio for recording - built into device
        t_nCurSel = m_AudioDevicesCtrl.GetCurSel();
        if ( t_nCurSel == CB_ERR )
        {
            strMessage.LoadString(IDS_ERR_MUSTSELECTAUDIODEVICE);
            MessageBox(strMessage);
            return;
        }
    }
}

```

```

    }
}

// If this is a new connection make sure that the label is unique
and valid
if ( m_bNewConnection )
{
    if ( !gm_Connections.IsLabelValid(m_Label) )
    {
        strMessage.LoadString(IDS_ERR_INVALIDLABEL);
        MessageBox(strMessage);
        return;
    }
}

// If a password is selected, make sure it is confirmed and 4
digits
if ( !m_Password.IsEmpty() )
{
    if ( m_Password.GetLength() < 4 )
    {
        strMessage.LoadString(IDS_ERR_PASSWORDLENGTH);
        MessageBox(strMessage);
        return;
    }
    if ( m_Password != m_Password2 )
    {
        strMessage.LoadString(IDS_ERR_PASSWORDMATCH);
        MessageBox(strMessage);
        return;
    }
}

CDialog::OnOK();
}

BOOL CConnectionDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    // TODO: Add your specialized code here and/or call the base
    class

    // if ( LOWORD(wParam) == IDC_CONNECTIONMETHOD &&
    // if ( HIWORD(wParam) == CBN_SELCHANGE )
    {
        UpdateData();
        UpdateConnectionInformationControls();
    }

    return CDialog::OnCommand(wParam, lParam);
}

void CConnectionDlg::OnCancel()
{
    if ( AfxMessageBox(IDS_CANCELAREYOUSURE, MB_YESNO |
    MB_ICONQUESTION) == IDNO )
        return;
}

```

```

        CDialog::OnCancel();
    }

void CConnectionDlg::OnAdvanced()
{
    CAdvConnectionDlg t_AdvConnectionDlg;

    t_AdvConnectionDlg.m_VideoPort = m_ConnectionInfo->m_SimpleVideo.m_dwPort;

    t_AdvConnectionDlg.m_VideoProfile = m_ConnectionInfo->m_SimpleVideo.m_dwProfile;

    if ( t_AdvConnectionDlg.DoModal() == IDOK )
    {
        m_ConnectionInfo->m_SimpleVideo.m_dwPort =
t_AdvConnectionDlg.m_VideoPort ;
        m_ConnectionInfo->m_SimpleVideo.m_dwProfile =
t_AdvConnectionDlg.m_VideoProfile ;
    }
}

#include "atlbase.h"
#import "..\Yellow Pages\VPD.ocx" named_guids no_namespace
void CConnectionDlg::OnBrowse()
{
    CComPtr<IVideoPeer> t_pYellowPages;
    CComBSTR t_bstrLabel ;
    CComBSTR t_bstrIPAddress;
    HRESULT t_hResult;
    _bstr_t t_bstr;
    CWnd* t_Wnd;

    t_hResult = t_pYellowPages.CoCreateInstance(CLSID_VideoPeer);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMSGAGETYPE_ERROR, "OnBrowse",
t_hResult);
        goto Exit1;
    }

    t_hResult = t_pYellowPages->raw_Lookup(lptList,
&t_bstrLabel.m_str, &t_bstrIPAddress.m_str);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMSGAGETYPE_ERROR, "OnBrowse",
t_hResult);
        goto Exit1;
    }

    t_pYellowPages.Release();

    t_bstr = t_bstrIPAddress;
    m_IPAddress.SetWindowText((char*) t_bstr );
    t_bstr = t_bstrLabel;
    t_Wnd = GetDlgItem(IDC_YELLOWPAGESENTRY) ;
}

```

```

        t_Wnd->SetWindowText((char*) t_bstr );

Exit1:
    return ;
}

void CConnectionDlg::OnMotiondetection()
{
    CMotionDetectionSettingsDlg t_MotionDetectionSettingsDlg;

    t_MotionDetectionSettingsDlg.m_Active = m_ConnectionInfo-
>m_SimpleVideo.m_MDAActive;
    t_MotionDetectionSettingsDlg.m_DwellTime = m_ConnectionInfo-
>m_SimpleVideo.m_MDDwellTime;
    t_MotionDetectionSettingsDlg.m_Sensitivity = m_ConnectionInfo-
>m_SimpleVideo.m_MDSensitivity;

    if ( t_MotionDetectionSettingsDlg.DoModal() == IDOK )
    {
        m_ConnectionInfo->m_SimpleVideo.m_MDAActive =
t_MotionDetectionSettingsDlg.m_Active ;
        m_ConnectionInfo->m_SimpleVideo.m_MDDwellTime =
t_MotionDetectionSettingsDlg.m_DwellTime ;
        m_ConnectionInfo->m_SimpleVideo.m_MDSensitivity =
t_MotionDetectionSettingsDlg.m_Sensitivity ;
    }
}
// ConnectionsFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "ConnectionsFrame.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;

////////////////////////////////////
/////
// CConnectionsFrame

IMPLEMENT_DYNCREATE(CConnectionsFrame, CMDIChildWnd)

CConnectionsFrame::CConnectionsFrame()
{
}

CConnectionsFrame::~CConnectionsFrame()
{
}

```

```

BEGIN_MESSAGE_MAP(CConnectionsFram , CMDIChildWnd)
//{{AFX_MSG_MAP(CConnectionsFrame)
    ON_WM_CREATE()
    ON_WM_CLOSE()
    ON_WM_DESTROY()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CConnectionsFrame message handlers

int CConnectionsFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_CONNECTIONS))
    {
        return FALSE;        // fail to create
    }

    return 0;
}

BOOL CConnectionsFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.x = gm_Layout.m_rectConnectionsWindow.left;
    cs.y = gm_Layout.m_rectConnectionsWindow.top;
    cs.cx = gm_Layout.m_rectConnectionsWindow.Width();
    cs.cy = gm_Layout.m_rectConnectionsWindow.Height();
    return CMDIChildWnd::PreCreateWindow(cs);
}

void CConnectionsFrame::OnClose()
{
    gm_Layout.m_bConnectionsWindowOpen = FALSE;

    CMDIChildWnd::OnClose();
}

void CConnectionsFrame::OnDestroy()
{
    CMDIChildWnd::OnDestroy();

    GetWindowRect(&gm_Layout.m_rectConnectionsWindow);
    GetParent()->ScreenToClient(&gm_Layout.m_rectConnectionsWindow);
}
// ConnectionsListView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nayal.h"

```

```

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "ConnectionsListView.h"
#include "ConnectionDlg.h"
#include "Main Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CConnectionsArray gm_Connections;
extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CConnectionsListView

IMPLEMENT_DYNCREATE(CConnectionsListView, CListView)

CConnectionsListView::CConnectionsListView()
{
}

CConnectionsListView::~CConnectionsListView()
{
}

BEGIN_MESSAGE_MAP(CConnectionsListView, CListView)
    //{{AFX_MSG_MAP(CConnectionsListView)
    ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLargeicons)
    ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallicons)
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
    ON_COMMAND(ID_ADD_CONNECTION, OnAddConnection)
    ON_COMMAND(ID_DELETE_CONNECTION, OnDeleteConnection)
    ON_UPDATE_COMMAND_UI(ID_DELETE_CONNECTION,
OnUpdateDeleteConnection)
    ON_NOTIFY_REFLECT(NM_DBLCLK, OnDbldclk)
    ON_COMMAND(ID_OPEN_CONNECTION, OnOpenConnection)
    ON_UPDATE_COMMAND_UI(ID_OPEN_CONNECTION, OnUpdateOpenConnection)
    ON_COMMAND(ID_CONNECT, OnConnect)
    ON_UPDATE_COMMAND_UI(ID_CONNECT, OnUpdateConnect)
    ON_COMMAND(ID_DISCONNECT, OnDisconnect)
    ON_UPDATE_COMMAND_UI(ID_DISCONNECT, OnUpdateDisconnect)
    ON_COMMAND(ID_EDIT_CONNECTION, OnEditConnection)
    ON_UPDATE_COMMAND_UI(ID_EDIT_CONNECTION, OnUpdateEditConnection)
    ON_WM_CONTEXTMENU()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
/////
// CConnectionsListView drawing

void CConnectionsListView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CConnectionsListView diagnostics

#ifdef _DEBUG
void CConnectionsListView::AssertValid() const
{
    CListView::AssertValid();
}

void CConnectionsListView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CConnectionsListView message handlers

BOOL CConnectionsListView::PreCreateWindow(CREATESTRUCT& cs)
{
    // Default to report view
    cs.style |= LVS_REPORT | LVS_NOSORTHEADER | LVS_SINGLESEL;

    return CListView::PreCreateWindow(cs);
}

void CConnectionsListView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    // Set up icons
    m_LargeImageList.Create(IDB_CONNECTIONSLARGEICONS, 32, 1,
    RGB(255, 255, 255));
    m_SmallImageList.Create(IDB_CONNECTIONSSMALLICONS, 16, 1,
    RGB(255, 255, 255));
    m_StateImageList.Create(IDB_CONNECTIONSSTATEICONS, 8, 1, RGB(255,
    0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

```

```

        // Set up columns
        t_strItem.LoadString(IDS_LABEL);
        t_ctlList.InsertColumn(0, t_strItem);
        SetColumnWidth(0, 110);
        t_strItem.LoadString(IDS_METHOD);
        t_ctlList.InsertColumn(1, t_strItem);
        SetColumnWidth(1, 160);
        t_strItem.LoadString(IDS_DATA);
        t_ctlList.InsertColumn(2, t_strItem);
        SetColumnWidth(2, 110);
        t_strItem.LoadString(IDS_TYPE);
        t_ctlList.InsertColumn(3, t_strItem);
        SetColumnWidth(3, 110);

        // Fill values
        UpdateListCtrl();
    }

void CConnectionsListView::OnViewSmallicons()
{
    if (GetViewType() != LVS_SMALLICON)
        SetViewType(LVS_SMALLICON);
}

void CConnectionsListView::OnViewDetails()
{
    if (GetViewType() != LVS_REPORT)
        SetViewType(LVS_REPORT);
}

void CConnectionsListView::OnViewLargeicons()
{
    if (GetViewType() != LVS_ICON)
        SetViewType(LVS_ICON);
}

void CConnectionsListView::OnViewList()
{
    if (GetViewType() != LVS_LIST)
        SetViewType(LVS_LIST);
}

BOOL CConnectionsListView::SetViewType(DWORD dwViewType)
{
    return(ModifyStyle(LVS_TYPEMASK, dwViewType & LVS_TYPEMASK));
}

DWORD CConnectionsListView::GetViewType()
{
    return(GetStyle() & LVS_TYPEMASK);
}

BOOL CConnectionsListView::SetColumnWidth(int Column, int Width)
{
    CListCtrl& t_ctlList = GetListCtrl();
    LVCOLUMN t_lvColumn;

```

```

        t_lvColumn.mask = LVCF_WIDTH;
        t_lvColumn.cx = Width;
        return t_ctlList.SetColumn(Column, &t_lvColumn);
    }

void CConnectionsListView::OnAddConnection()
{
    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONCONNECTION) )
        return ;

    gm_Connections.AddConnection();
}

void CConnectionsListView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
        case NUPDATE_NEWCONNECTION:

        case NUPDATE_DELETECONNECTION:
        case NUPDATE_UPDATECONNECTION:
        case NUPDATE_OPENPROFILE:
            UpdateListCtrl();
            break;
    };
}

bool CConnectionsListView::UpdateListCtrl()
{
    bool t_bReturn = false;
    int i;
    CString t_str;
    CListCtrl& t_ctlList = GetListCtrl();

    // Clear contents of control
    t_ctlList.DeleteAllItems();

    // Get array of connections
    for ( i = 0; i < gm_Connections.GetSize(); i++ )
    {
        CConnectionInfo& t_ConnectionInfo = gm_Connections[i];
        int t_nIcon = 0;

        switch ( t_ConnectionInfo.m_ConnectionType )
        {
            case CONTYPE_VIDEOCONFERENCE:
                t_nIcon = 4;
                break;
            case CONTYPE_INSTANTMESSENGER:
                switch ( t_ConnectionInfo.m_ConnectionMethod )
                {
                    case CONNECTION_LOCALTCPIP:
                        t_nIcon = 2;

```

```

        break;
    case CONNECTION_IPADDRESS:
    case CONNECTION_YELLOWPAGES:
        t_nIcon = 3;
        break;
    };
    break;
case CONTYPE_VIDEOSURVEILLANCE:
    switch ( t_ConnectionInfo.m_ConnectionMethod )
    {
    case CONNECTION_LOCALTCPIP:
        t_nIcon = 0;
        break;
    case CONNECTION_IPADDRESS:
    case CONNECTION_YELLOWPAGES:
        t_nIcon = 1;
        break;
    };
    break;
}

// Insert item
t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
                    i,
t_ConnectionInfo.m_Label,
                    NULL, NULL, t_nIcon,
                    NULL);

// Set the remaining fields
t_ConnectionInfo.GetConnectionMethodString(t_str);
t_ctlList.SetItemText(i, 1, t_str);
t_ConnectionInfo.GetConnectionKeyData(t_str);
t_ctlList.SetItemText(i, 2, t_str);
t_ConnectionInfo.GetConnectionTypeString(t_str);
t_ctlList.SetItemText(i, 3, t_str);
}

t_ctlList.SetItemState(0, LVIS_SELECTED, LVIS_SELECTED );

t_bReturn = true;
//Exit1:
return t_bReturn;
}

bool CConnectionsListView::GetSelectedConnectionLabel(CString&
strLabel)
{
    bool t_bReturn = false;

    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem ;

    // Get selected item - this is a single-selection list control
    t_nItem = GetSelectedConnectionItem();
    if ( t_nItem == -1 )
        goto Exit1;

```

```

        // Get the m_Connections array index for that item
        strLabel = t_ctlList.GetItemText(t_nItem, 0);

        t_bReturn = true;
Exit1:
        return t_bReturn;
    }

void CConnectionsListView::OnDeleteConnection()
{
    CString t_strLabel;

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONCONNECTION) )
        return ;

    // Get label of selected item and delete it
    if ( GetSelectedConnectionLabel(t_strLabel) )
        gm_Connections.DeleteConnection(t_strLabel);
}

void CConnectionsListView::OnUpdateDeleteConnection(CCmdUI* pCmdUI)
{
    if ( GetSelectedConnectionItem(false) == -1 )
        pCmdUI->Enable(FALSE);
}

void CConnectionsListView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    switch ( gm_AppConfig.m_nOnDoubleClickConnectionsItem )
    {
        case ONDBLCLICKCONNECTIONITEM_CONFIGCONNECTION:
            OnEditConnection();
            break;
        case ONDBLCLICKCONNECTIONITEM_OPENCONNECTION:
            OnOpenConnection();
            break;
        default:
            break;
    }

    *pResult = 0;
}

void CConnectionsListView::OnOpenConnection()
{
    CProjectNalayApp* t_ProjectNalayApp;
    CString t_strLabel;

    t_ProjectNalayApp = (CProjectNalayApp*) AfxGetApp();
    // Get label of currently selectd connection
    if ( GetSelectedConnectionLabel(t_strLabel) )
        t_ProjectNalayApp->OpenConnectionWindow(t_strLabel);
}

void CConnectionsListView::OnUpdateOpenConnection(CCmdUI* pCmdUI)
{

```

```

        if ( GetSelectedConnectionItem(false) == -1 )
            pCmdUI->Enable(FALSE);
    }

int CConnectionsListView::GetSelectedConnectionItem(bool bFeedback)
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem = -1;

    // Get selected item - this is a single-selection list control
    POSITION t_pos = t_ctlList.GetFirstSelectedItemPosition();

    // Validate that an item was selected
    if (t_pos == NULL)
    {
        if ( bFeedback )
            NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_WARNING, "",
IDS_NOCONNECTIONSELECTED);
        goto Exit1;
    }

    // Get the item number selected
    t_nItem = t_ctlList.GetNextSelectedItem(t_pos);

Exit1:
    return t_nItem;
}

void CConnectionsListView::OnConnect()
{
    int t_nItem;
    int t_nIndex;
    CListCtrl& t_ctlList = GetListCtrl();

    // Get the item that is selected
    // This option should not even be available if no item is
selected
    t_nItem = GetSelectedConnectionItem();
    if ( t_nItem == -1 )
        goto Exit1;

    // Get the m_Connections array index for that item
    t_nIndex = (int) t_ctlList.GetItemData(t_nItem);

    // Connect
    gm_Connections.Connect(t_nIndex );

    // Set the state to checked
    // t_ctlList.SetItemState(t_nItem, INDEXTOSTATEIMAGEMASK(1),
LVIS_STATEIMAGEMASK);

Exit1:
    return;
}

void CConnectionsListView::OnUpdateConnect(CCmdUI* pCmdUI)
{

```

```

        if ( GetSelectedConnectionItem(false) == -1 )
            pCmdUI->Enable(FALSE);
    }

void CConnectionsListView::OnDisconnect()
{
    CProjectNalayApp* t_ProjectNalayApp;
    CString t_strLabel;

    t_ProjectNalayApp = (CProjectNalayApp*) AfxGetApp();
    // Get label of currently selectd connection
    if ( GetSelectedConnectionLabel(t_strLabel) )
        t_ProjectNalayApp->CloseConnectionWindow(t_strLabel);

    // Set the state to non-checked, ie.e blank
    // t_ctlList.SetItemState(t_nItem, INDEXTOSTATEIMAGEMASK(2),
    LVIS_STATEIMAGEMASK);

    return;
}

void CConnectionsListView::OnUpdateDisconnect(CCmdUI* pCmdUI)
{
    if ( GetSelectedConnectionItem(false) == -1 )
        pCmdUI->Enable(FALSE);
}

void CConnectionsListView::OnEditConnection()
{
    CString t_strLabel;

    // Check password
    if ( !PromptSecurityPassword(PWD_PROMPT_ONCONNECTION) )
        return ;

    // Get label of selected item and delete it
    if ( GetSelectedConnectionLabel(t_strLabel) )
        gm_Connections.EditConnection(t_strLabel);
}

void CConnectionsListView::OnUpdateEditConnection(CCmdUI* pCmdUI)
{
    if ( GetSelectedConnectionItem(false) == -1 )
        pCmdUI->Enable(FALSE);
}

void CConnectionsListView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    DisplayContextMenu(this, point, IDR_POPUP_CONNECTIONS);
}
// ConnectionsView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "ConnectionsView.h"

```

```

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CConnectionsView

IMPLEMENT_DYNCREATE(CConnectionsView, CFormView)

CConnectionsView::CConnectionsView()
    : CFormView(CConnectionsView::IDD)
{
    //{AFX_DATA_INIT(CConnectionsView)
    // NOTE: the ClassWizard will add member initialization
here
    //}}AFX_DATA_INIT
}

CConnectionsView::~CConnectionsView()
{
}

void CConnectionsView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CConnectionsView)
    DDX_Control(pDX, IDC_CONNECTIONSLIST, m_ConnectionsList);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CConnectionsView, CFormView)
    //{AFX_MSG_MAP(CConnectionsView)
    // NOTE - the ClassWizard will add and remove mapping
macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CConnectionsView diagnostics

#ifdef _DEBUG
void CConnectionsView::AssertValid() const
{
    CFormView::AssertValid();
}

void CConnectionsView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}
#endif // _DEBUG

```

```

////////////////////////////////////
////////
// CConnectionsView message handlers

#include "stdafx.h"
#include "CX10.h"

#include "objbase.h"
#include "atlbase.h"

CX10::CX10()
{
    m_pCM17 = NULL;
    m_nPort = 1;
}

bool CX10::Initialize()
{
    bool t_bReturn = false;
    short t_nResult;
    HRESULT t_hResult;

    t_hResult = CoCreateInstance( CLSID_controlcm, NULL,
                                CLSCTX_INPROC_SERVER,
                                IID_controlcm,
                                (LPVOID*) &m_pCM17 );

    if FAILED(t_hResult)
    {
        // NSENDFEEDBACKMESSAGE
        goto Exit1;
    }

    m_pCM17->comport = m_nPort ;

    t_nResult = m_pCM17->Init();
    if ( t_nResult != 0 )
    {
        // NSENDFEEDBACKMESSAGE
        goto Exit1;
    }

    t_bReturn = true;
Exit1:
    if ( !t_bReturn ) Uninitialize();
    return t_bReturn;
}

bool CX10::Uninitialize()
{
    if ( m_pCM17 != NULL )
        m_pCM17->Release();

    m_pCM17 = NULL;
    return true;
}

```

```

bool CX10::SetPort(short nPort)
{
    m_nPort = nPort;
    return true;
}

bool CX10::ExecuteCommand(LPSTR szHouseCode, LPSTR szDeviceCode, short
Command, short Brightness)
{
    bool t_bReturn = true;
    CComBSTR t_bstrHouseCode;
    CComBSTR t_bstrDeviceCode;
    short t_Command;
    short t_Brightness;

    t_bstrHouseCode = szHouseCode;
    t_bstrDeviceCode = szDeviceCode;
    t_Command = Command;
    t_Brightness = Brightness;

    m_pCM17->Exec(&t_bstrHouseCode.m_str, &t_bstrDeviceCode.m_str,
&t_Command, &t_Brightness);

    return t_bReturn;
}

bool CX10::ExecuteCommand(int nHouseCode, int nDeviceCode, short
Command, short Brightness)
{
    char t_szHouseCode[2];
    char t_szDeviceCode[40];

    t_szHouseCode[0] = 'A';
    t_szHouseCode[0] += (char) nHouseCode;
    t_szHouseCode[1] = NULL;

    itoa(nDeviceCode, t_szDeviceCode, 10);

    return ExecuteCommand(t_szHouseCode, t_szDeviceCode, Command,
Brightness);
}
//-----
// File: DirectPlay Info Classes.cpp
//
// Desc: DirectPlay-related classes for managing individual and groups
of connections filter for detecting motion in a video stream
//
// Comments:
//
// Debug Notes:
//
// History: 03/22/02    LCK            Created
//
// Copyright (c) 2002 BKLK Inc. All rights reserved.

```

```

//-----
-----

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"
#include "ConnectionDlg.h"
#include "EventDialog.h"
#include "EMailDlg.h"
#include "PhoneDlg.h"
#include "VideoRecordDlg.h"
#include "X10Dlg.h"
#include "AudioDlg.h"

#include "SimpleMAPI.h"
#include "CX10.h"
#include "TAPIControl.h"

#include "SimpleDirectAudio.h"

extern CLayout gm_Layout;
extern CAppConfig gm_AppConfig;
extern CRITICAL_SECTION gm_csEMailAction;
extern CRITICAL_SECTION gm_csAudioAction;
extern CRITICAL_SECTION gm_csPhoneAction;
extern CRITICAL_SECTION gm_csX10Action;

IMPLEMENT_SERIAL( CEventInfo, CObject, 1 )
#define VERSION_CONNECTIONSFILE 4

bool CConnectionsArray::AddConnection()
{
    bool t_bReturn = false;
    CConnectionDlg t_ConnectionDlg;
    CConnectionInfo t_ConnectionInfo;

    // Prompt user
    t_ConnectionDlg.m_ConnectionInfo = &t_ConnectionInfo;
    if ( t_ConnectionDlg.DoModal() == IDOK )
    {
        // Add the connection
        Add(t_ConnectionInfo);

        // Update everyone
        NUpdateAllViews(NULL, NUPDATE_NEWCONNECTION, (CObject*)
NULL);

        t_bReturn = true;
    }

    return t_bReturn;
}

int CConnectionsArray::GetIndexFromLabel(CString strLabel)
{

```

```

    int t_nIndex = -1;

    // Search through connections until there is a label match
    for (int i = 0; i < GetSize(); i++)
    {
        if ( ElementAt(i).m_Label == strLabel )
            t_nIndex = i;
    }

    // If there is no label match, generate an error
    // Had to remove this because being used for toolbar checks
    // if ( t_nIndex == -1 )
    //     NSENDFEEDBACKIDS(FEEDBACKMESSAGE_TYPE_ERROR, "",
    IDS_NOMATCHINGINDEX);

    return t_nIndex;
}

bool CConnectionsArray::DeleteConnection(CString strLabel)
{
    bool t_bReturn = false;
    int t_nIndex;

    // If connection is open, do not delete
    if ( gm_Layout.GetLayoutConnectionIndexFromLabel(strLabel) >= 0 )
    {
        AfxMessageBox(IDS_ERR_CANNOTDELETEOPENCONNECTION);
        goto Exit1;
    }

    // Validate index
    t_nIndex = GetIndexFromLabel(strLabel);
    if ( t_nIndex < 0 )
        goto Exit1;

    // Send the update with the index of the about-to-be deleted
    connection
    // in case there is some clean up work
    NUpdateAllViews(NULL, NUPDATE_PREDELETECONNECTION, (CObject*)
    NULL );

    // ONLY Then delete the ConnectionInfo
    RemoveAt(t_nIndex);

    // Send the update with the index of the newly deleted connection
    // in case there is some post clean up work
    NUpdateAllViews(NULL, NUPDATE_DELETECONNECTION, (CObject*) NULL
    );

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

bool CConnectionsArray::EditConnection(CString strLabel)
{
    bool t_bReturn = false;

```

```

int t_nIndex;
CConnectionDlg t_ConnectionDlg;
CConnectionInfo t_ConnectionInfo;

t_nIndex = GetIndexFromLabel(strLabel);

// Validate index
if ( t_nIndex < 0 )
    goto Exit1;

// Set connection info for dialog
ElementAt(t_nIndex).GetCopy(t_ConnectionInfo);
t_ConnectionDlg.m_ConnectionInfo = &t_ConnectionInfo;
t_ConnectionDlg.m_bNewConnection = false;
if ( t_ConnectionDlg.DoModal() == IDOK )
{
//      ElementAt(t_nIndex).SetCopy(t_ConnectionInfo);
//      SetAt(t_nIndex, t_ConnectionInfo);

// Send the update with the index of the updated connection
// in case there is some clean up work
NUpdateAllViews(NULL, NUPDATE_UPDATECONNECTION, (CObject*)
NULL );
}

t_bReturn = true;
Exit1:
return t_bReturn;
}

bool CConnectionsArray::IsValidIndex(int nIndex)
{
    // Validate the index
    if ( nIndex < 0 || nIndex >= GetSize() )
        return false;

    return true;
}

bool CConnectionsArray::IsLabelValid(CString strLabel)
{
    bool t_bReturn = false;
    int i;

    // Empty label is invalid
    if ( strLabel.IsEmpty() )
        goto Exit1;

    // Check if there is a match
    for ( i = 0; i < GetSize(); i++ )
        if ( ElementAt(i).m_Label == strLabel )
            break;

    // No match - label is valid
    if ( i == GetSize() )
        t_bReturn = true;
}

```

```

Exit1:
    return t_bReturn;
}

bool CEventsArray::IsLabelValid(CString strLabel)
{
    bool t_bReturn = false;
    int i;

    // Empty label is invalid
    if ( strLabel.IsEmpty() )
        goto Exit1;

    // Check if there is a match
    for ( i = 0; i < GetSize(); i++ )
        if ( ElementAt(i).m_Label == strLabel )
            break;

    // No match - label is valid
    if ( i == GetSize() )
        t_bReturn = true;

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::IsConnectionEstablished(CString strLabel)
{
    bool t_bReturn = false;
    int t_nIndex ;

    // First get index
    t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn = ElementAt(t_nIndex).IsConnectionEstablished();

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::Connect(CString strLabel, HWND hWnd)
{
    bool t_bReturn = false;
    int t_nIndex ;

    // Check if any other connections are already active
    for ( int i = 0; i < GetSize(); i++ )
    {
        if ( ElementAt(i).IsConnectionEstablished() )
        {
            NSEDFEEDBACKIDS(FEEDBACKMESSAGETYPE_ERROR, strLabel,
IDS_ERR_CONNECTIONESTABLISHED);

```

```

        goto Exit1;
    }
}

// First get index
t_nIndex = GetIndexFromLabel(strLabel);

// Check index
if ( t_nIndex == -1 )
    goto Exit1;

// Let object connect itself
t_bReturn = ElementAt(t_nIndex).Connect(hWnd);

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::SendEvents(CString strLabel)
{
    bool t_bReturn = false;
    int t_nIndex ;

    // First get index
    t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn = ElementAt(t_nIndex).SendEvents();

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::ReceiveEvents(CString strLabel, CConnectionMsg*
pConnectionMsg)
{
    bool t_bReturn = false;
    int t_nIndex ;

    // First get index
    t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn = ElementAt(t_nIndex).ReceiveEvents(pConnectionMsg);

Exit1:
    return t_bReturn;
}

```

```

bool CConnectionsArray::Disconnect(CString strLabel)
{
    bool t_bReturn = false;

    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn = ElementAt(t_nIndex).Disconnect();

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::GetConnectionType(CString strLabel, UINT&
ConnectionType)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        return false;

    // Update the connection type argument
    ConnectionType = ElementAt(t_nIndex).m_ConnectionType;
    return true;
}

bool CConnectionsArray::GetVideoDevice(CString strLabel, CString&
strVideoDevice)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        return false;

    // Update the connection type argument
    strVideoDevice =
ElementAt(t_nIndex).m_SimpleVideo.m_strVideoDevice;
    return true;
}

bool CConnectionsArray::SerializeEvents(CString strLabel, CArchive&
archive)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )

```

```

        return false;

        // Update the connection type argument
        ElementAt(t_nIndex).m_Events.Serialize(archive);

        return true;
    }

bool CConnectionsArray::GetIPAddress(CString strLabel, CString&
strIPAddress)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        return false;

    // Update the connection type argument
    return ElementAt(t_nIndex).GetIPAddress(strIPAddress);
}

bool CConnectionsArray::GetPort(CString strLabel, DWORD& dwPort)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        return false;

    // Update the connection type argument
    dwPort = ElementAt(t_nIndex).m_SimpleVideo.m_dwPort;
    return true;
}

bool CConnectionsArray::GetAudioDevice(CString strLabel, CString&
strAudioDevice)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        return false;

    // Update the connection type argument
    strAudioDevice =
    ElementAt(t_nIndex).m_SimpleVideo.m_strAudioDevice;
    return true;
}

bool CConnectionsArray::GetConnectionMethod(CString strLabel, UINT&
ConnectionMethod)
{
    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

```

```

        // Check index
        if ( t_nIndex == -1 )
            return false;

        // Update the connection type argument
        ConnectionMethod = ElementAt(t_nIndex).m_ConnectionMethod;
        return true;
    }

bool CConnectionsArray::SendMessage(CString strLabel, CConnectionMsg&
ConnectionMsg)
{
    bool t_bReturn = false;

    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn =
ElementAt(t_nIndex).m_SimpleDirectPlay.SendMessage(ConnectionMsg);

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::SendMessage(CString strLabel, CString
strMessage)
{
    bool t_bReturn = false;

    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn =
ElementAt(t_nIndex).m_SimpleDirectPlay.SendMessage(strMessage);

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::TriggerScheduledEvent(CTime t_CurrentTime)
{
    for ( int i = 0; i < GetSize(); i++ )
        ElementAt(i).TriggerScheduledEvent(t_CurrentTime);

    return true;
}

```

```

bool CConnectionsArray::TriggerAlarm(CString strLabel)
{
    bool t_bReturn = false;

    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object connect itself
    t_bReturn = ElementAt(t_nIndex).TriggerAlarm();

Exit1:
    return t_bReturn;
}

bool CEventInfo::TriggerScheduledEvent(CTime t_CurrentTime,
CConnectionInfo* pConnectionInfo)
{
    int t_CurHour = t_CurrentTime.GetHour();
    int t_EventHour = m_StartTime.GetHour();
    int t_CurMinute = t_CurrentTime.GetMinute();
    int t_EventMinute = m_StartTime.GetMinute();
    int t_CurDayOfWeek = t_CurrentTime.GetDayOfWeek();

    // Check if this is an alarm
    if ( m_EventType != EVENTTYPE_SCHEDULEDEVENT )
        return false;

    // Check if day of week is a match
    switch ( t_CurDayOfWeek )
    {
    case 1:
        if ( !m_Sunday ) return false;
        break;
    case 2:
        if ( !m_Monday ) return false;
        break;
    case 3:
        if ( !m_Tuesday ) return false;
        break;
    case 4:
        if ( !m_Wednesday ) return false;
        break;
    case 5:
        if ( !m_Thursday ) return false;
        break;
    case 6:
        if ( !m_Friday ) return false;
        break;
    case 7:
        if ( !m_Saturday ) return false;
        break;
    }
}

```

```

// Check is hour/min is a match
if ( t_CurHour == t_EventHour &&
     t_CurMinute == t_EventMinute )
{
    CString t_strMessage;
    t_strMessage.LoadString(IDS_EVENTTRIGGERED);
    NSENDERFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_STATUS,
t_strMessage, m_Label);
    m_Actions.Trigger(pConnectionInfo, this);
}

return true;
}

bool CEventInfo::TriggerAlarm(CConnectionInfo* pConnectionInfo)
{
    CTime t_CurrentTime = CTime::GetCurrentTime();
    int t_CurHour = t_CurrentTime.GetHour();
    int t_CurMin = t_CurrentTime.GetMinute();
    int t_CurDayOfWeek = t_CurrentTime.GetDayOfWeek();
    CString t_strMessage;

    // Check if this is an alarm
    if ( m_EventType != EVENTTYPE_ALARM )
        return false;

    // Check if day of week is a match
    switch ( t_CurDayOfWeek )
    {
    case 1:
        if ( !m_Sunday ) return false;
        break;
    case 2:
        if ( !m_Monday ) return false;
        break;
    case 3:
        if ( !m_Tuesday ) return false;
        break;
    case 4:
        if ( !m_Wednesday ) return false;
        break;
    case 5:
        if ( !m_Thursday ) return false;
        break;
    case 6:
        if ( !m_Friday ) return false;
        break;
    case 7:
        if ( !m_Saturday ) return false;
        break;
    }

    // Check is hour/min is in correct range
    if ( t_CurHour < m_StartTime.GetHour() )
        return false;
    if ( t_CurHour == m_StartTime.GetHour() )

```

```

        if ( t_CurMin < m_StartTime.GetMinute() )
            return false;
        if ( t_CurHour > m_EndTime.GetHour() )
            return false;
        if ( t_CurHour == m_EndTime.GetHour() )
            if ( t_CurMin > m_EndTime.GetMinute() )
                return false;

        t_strMessage.LoadString(IDS_ALARMTRIGGERED);
        NSENDMESSAGE(FEEDBACKMESSAGE_STATUS, t_strMessage,
m_Label);
        m_Actions.Trigger(pConnectionInfo, this);

        return true;
    }

bool CEventsArray::TriggerScheduledEvent(CTime t_CurrentTime,
CConnectionInfo* pConnectionInfo)
{
    for ( int i = 0; i < GetSize(); i++ )
        ElementAt(i).TriggerScheduledEvent(t_CurrentTime,
pConnectionInfo);

    return true;
}

bool CActionsArray::Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo)
{
    for ( int i = 0; i < GetSize(); i++ )
        ((CActionInfo*) ElementAt(i))->Trigger(pConnectionInfo,
pEventInfo);

    return true;
}

bool CEmailAction::Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo)
{
    CString t_strMessage;
    CString t_strDescription;

    t_strMessage.LoadString(IDS_EMAILACTIONTRIGGERED);
    GetActionSummary(t_strDescription);
    NSENDMESSAGE(FEEDBACKMESSAGE_STATUS, t_strMessage,
t_strDescription);

    return ExecuteAction(pConnectionInfo);
}

bool CAudioAction::Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo)
{
    CString t_strMessage;
    CString t_strDescription;

```

```

        t_strMessage.LoadString(IDS_AUDIOACTIONTRIGGERED);
        GetActionSummary(t_strDescription);
        NSENDERFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_STATUS, t_strMessage,
t_strDescription);

        return ExecuteAction();
    }

bool CPhoneAction::Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo)
{
    CString t_strMessage;
    CString t_strDescription;

    t_strMessage.LoadString(IDS_PHONEACTIONTRIGGERED);
    GetActionSummary(t_strDescription);
    NSENDERFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_STATUS, t_strMessage,
t_strDescription);

    return ExecuteAction();
}

bool CVideoRecordAction::Trigger(CConnectionInfo* pConnectionInfo,
CEventInfo* pEventInfo)
{
    CString t_strMessage;
    CString t_strDescription;

    t_strMessage.LoadString(IDS_VIDEORECORDACTIONTRIGGERED);
    GetActionSummary(t_strDescription);
    NSENDERFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_STATUS, t_strMessage,
t_strDescription);

    return ExecuteAction(pConnectionInfo, pEventInfo);
}

bool CX10Action::Trigger(CConnectionInfo* pConnectionInfo, CEventInfo*
pEventInfo)
{
    CString t_strMessage;
    CString t_strDescription;

    t_strMessage.LoadString(IDS_X10ACTIONTRIGGERED);
    GetActionSummary(t_strDescription);
    NSENDERFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_STATUS, t_strMessage,
t_strDescription);

    return ExecuteAction();
}

void CConnectionsArray::Serialize( CArchive& archive )
{
    //    CArray<CConnectionInfo, CConnectionInfo&>::Serialize( archive );
    int t_nVersion;
    CString t_strIdentifier;
    int t_nSize;

```

```

    t_strIdentifier.LoadString(IDS_CONNECTIONSFILEIDENTIFIER);
    t_nVersion = VERSION_CONNECTIONSFILE;

// now do the stuff for our specific class
if( archive.IsStoring() )
{
    // First place identifier and version number
    archive << t_strIdentifier;
    archive << t_nVersion ;

    // Continue with the rest of the serialization
    archive << GetSize();
    for ( int i = 0; i < GetSize(); i++ )
        ElementAt(i).Serialize ( archive );
}
else
{
    CConnectionInfo t_ConnectionInfo;
    CString t_strIdentifierRead;
    int t_nVersionRead;

    // First read identifier
    archive >> t_strIdentifierRead;

    // Check to see if it the correct file type
    if ( t_strIdentifierRead != t_strIdentifier )
        return;

    // Check to see if it is the correct version
    archive >> t_nVersionRead;
    if ( t_nVersionRead != t_nVersion )
        return;

    // Clear out current array contents
    RemoveAll();

    // Read in new contents
    archive >> t_nSize;
    for ( int i = 0; i < t_nSize; i++ )
    {
        t_ConnectionInfo.Serialize (archive );
        Add ( t_ConnectionInfo );
    }
}

void CConnectionsArray::Debug()
{
    CConnectionInfo t_ConnectionInfo;

    for (int i = 0; i < GetSize(); i++ )
    {
        ElementAt(i).GetCopy(t_ConnectionInfo);
    }
}

CAudioAction::CAudioAction()

```

```

{
    m_ActionType = ACTIONTYPE_AUDIO;
}

CEMailAction::CEMailAction()
{
    m_ActionType = ACTIONTYPE_EMAIL;
    m_AttachVideo = FALSE;
    m_Duration = gm_AppConfig.m_DefaultVideoRecordDuration;
    m_KeepLeadingVideo = TRUE;
}

CActionsArray::CActionsArray()
{
}

CEventInfo::CEventInfo()
{
    m_EventType = EVENTTYPE_SCHEDULEDEVENT;
    // m_StartTime = CTime::GetCurrentTime();
    // m_EndTime = CTime::GetCurrentTime();
    m_StartTime = CTime(2002, 01, 01, 9, 0, 0);
    m_EndTime = CTime(2002, 01, 01, 17, 0, 0);

    m_Sunday = 0;
    m_Monday= TRUE;
    m_Tuesday= TRUE;
    m_Wednesday= TRUE;
    m_Thursday= TRUE;
    m_Friday= TRUE;
    m_Saturday= 0;
}

CEventsArray::CEventsArray()
{
}

CActionInfo::CActionInfo()
{
}

bool CConnectionsArray::AddEvent(CString strLabel, int nEventType)
{
    bool t_bReturn = false;

    // First get index
    int t_nIndex = GetIndexFromLabel(strLabel);

    // Check index
    if ( t_nIndex == -1 )
        goto Exit1;

    // Let object add new event by itself
    t_bReturn = ElementAt(t_nIndex).AddEvent(nEventType);
}

```

```

Exit1:
    return t_bReturn;
}

CEventInfo& CEventInfo::operator= ( CEventInfo& EventInfo)
{
    m_EventType = EventInfo.m_EventType ;
    m_Label= EventInfo.m_Label;
    m_StartTime= EventInfo.m_StartTime;
    m_EndTime= EventInfo.m_EndTime;
    m_Sunday= EventInfo.m_Sunday;
    m_Monday= EventInfo.m_Monday;
    m_Tuesday= EventInfo.m_Tuesday;
    m_Wednesday= EventInfo.m_Wednesday;
    m_Thursday= EventInfo.m_Thursday;
    m_Friday= EventInfo.m_Friday;
    m_Saturday= EventInfo.m_Saturday;

    m_Actions.DeleteAll();

    for (int i = 0; i < EventInfo.m_Actions.GetSize(); i++ )
    {
        CEmailAction * t_EmailAction;
        CPhoneAction * t_PhoneAction;
        CVideoRecordAction * t_VideoRecordAction;
        CX10Action * t_X10Action;
        CAudioAction * t_AudioAction;

        CActionInfo * t_ActionInfo = ( CActionInfo * )
EventInfo.m_Actions.ElementAt(i);
        switch ( t_ActionInfo->m_ActionType )
        {
            case ACTIONTYPE_EMAIL:
                t_EmailAction = new CEmailAction( (CEmailAction * )
t_ActionInfo );
                m_Actions.Add( t_EmailAction );
                break;
            case ACTIONTYPE_PHONE:
                t_PhoneAction = new CPhoneAction ( (CPhoneAction * )
t_ActionInfo );
                m_Actions.Add( t_PhoneAction );
                break;
            case ACTIONTYPE_RECORDVIDEO:
                t_VideoRecordAction = new CVideoRecordAction (
(CVideoRecordAction * ) t_ActionInfo );
                m_Actions.Add( t_VideoRecordAction );
                break;
            case ACTIONTYPE_X10:
                t_X10Action = new CX10Action ( (CX10Action * )
t_ActionInfo );
                m_Actions.Add( t_X10Action );
                break;
            case ACTIONTYPE_AUDIO:
                t_AudioAction = new CAudioAction ( (CAudioAction * )
t_ActionInfo );
                m_Actions.Add( t_AudioAction );
                break;

```

```

    }
}

return EventInfo;
}

bool CEventInfo::GetEventTypeString(CString& strEventType)
{
    bool t_bReturn = true;

    switch ( m_EventType )
    {
    case EVENTTYPE_ALARM:
        strEventType.LoadString(IDS_ALARM);
        break;
    case EVENTTYPE_SCHEDULEDEVENT:
        strEventType.LoadString(IDS_SCHEDULEDEVENT);
        break;
    default:
        t_bReturn = false;
    }

    return t_bReturn;
}

bool CEventInfo::GetEventDetailString(CString& t_str)
{
    CString t_strDays;

    if ( m_Sunday ) t_strDays += "Sun";
    if ( m_Monday ) t_strDays += "Mon";
    if ( m_Tuesday ) t_strDays += "Tue";
    if ( m_Wednesday ) t_strDays += "Wed";
    if ( m_Thursday ) t_strDays += "Thur";
    if ( m_Friday ) t_strDays += "Fri";
    if ( m_Saturday ) t_strDays += "Sat";

    t_str.Format("Day(s):%s, Actions: %d", (LPCTSTR) t_strDays,
m_Actions.GetSize() );

    return true;
}

bool CConnectionsArray::EditEvent(CString strConnectionLabel, CString
strEventLabel)
{
    bool t_bReturn = false;
    int t_nConnectionIndex = -1;

    // First get connection index
    t_nConnectionIndex = GetIndexFromLabel(strConnectionLabel);

    // Validate the index
    if ( t_nConnectionIndex < 0 )
        goto Exit1;
}

```

```

        t_bReturn =
ElementAt(t_nConnectionIndex).m_Events.EditEvent(strConnectionLabel,
strEventLabel);

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::IsEventLabelValid(CString strConnectionLabel,
CString strEventLabel)
{
    bool t_bReturn = false;
    int t_nConnectionIndex = -1;

    // First get connection index
    t_nConnectionIndex = GetIndexFromLabel(strConnectionLabel);

    // Validate the index
    if ( t_nConnectionIndex < 0 )
        goto Exit1;

    t_bReturn =
ElementAt(t_nConnectionIndex).IsEventLabelValid(strEventLabel);

Exit1:
    return t_bReturn;
}

bool CConnectionsArray::DeleteEvent(CString strConnectionLabel, CString
strEventLabel)
{
    bool t_bReturn = false;
    int t_nConnectionIndex = -1;

    // First get connection index
    t_nConnectionIndex = GetIndexFromLabel(strConnectionLabel);

    // Validate the index
    if ( t_nConnectionIndex < 0 )
        goto Exit1;

    t_bReturn =
ElementAt(t_nConnectionIndex).m_Events.DeleteEvent(strEventLabel);

Exit1:
    return t_bReturn;
}

bool CEventsArray::EditEvent(CString strConnectionLabel, CString
strEventLabel)
{
    bool t_bReturn = false;
    int t_nEventsIndex = -1;
    CEventDialog t_EventDialog;
    CEventInfo t_EventInfo;

    // First get connection index

```

```

    t_nEventsIndex = GetIndexFromLabel(strEventLabel);

    // Validate the index
    if ( t_nEventsIndex < 0 )
        goto Exit1;

    // Copy the EventInfo to dialog's EventInfo
    t_EventDialog.m_EventInfo = ElementAt(t_nEventsIndex );
    t_EventDialog.m_ConnectionLabel = strConnectionLabel;
    t_EventDialog.m_bNewEvent = false;

    if ( t_EventDialog.DoModal() == IDOK )
    {
        ElementAt(t_nEventsIndex ) = t_EventDialog.m_EventInfo;
        t_bReturn = true;
    }

Exit1:
    return t_bReturn;
}

bool CEventsArray::DeleteEvent(CString strEventLabel)
{
    bool t_bReturn = false;
    int t_nEventsIndex = -1;

    // First get connection index
    t_nEventsIndex = GetIndexFromLabel(strEventLabel);

    // Validate the index
    if ( t_nEventsIndex < 0 )
        goto Exit1;

    // Copy the EventInfo to dialog's EventInfo
    RemoveAt(t_nEventsIndex );

    t_bReturn = true;

Exit1:
    return t_bReturn;
}

int CEventsArray::GetIndexFromLabel(CString strEventLabel)
{
    int t_nEventIndex = -1;

    // Search through connections until there is a label match
    for (int i = 0; i < GetSize(); i++ )
    {
        if ( ElementAt(i).m_Label == strEventLabel )
            t_nEventIndex = i;
    }

    // If there is no label match, generate an error
    if ( t_nEventIndex == -1 )
        NSENDFEEDBACKIDS(FEEDBACKMESSAGE_TYPE_ERROR, "",
IDS_ERR_NOMATCHINGEVENTINDEX);
}

```

```

        return t_nEventIndex ;
    }

void CEventsArray::Serialize( CArchive& archive )
{
    int t_nSize;

    if( archive.IsStoring() )
    {
        archive << GetSize();
        for ( int i = 0; i < GetSize(); i++ )
            ElementAt(i).Serialize ( archive );
    }
    else
    {
        CEventInfo t_EventInfo;

        // Clear out current array contents
        RemoveAll();

        // Read in new contents
        archive >> t_nSize;
        for ( int i = 0; i < t_nSize; i++ )
        {
            t_EventInfo.Serialize ( archive );
            Add ( t_EventInfo );
        }
    }
}

void CEventInfo::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
    {
        archive << m_EventType;
        archive << m_Label;
        archive << m_StartTime;
        archive << m_EndTime;
        archive << m_Sunday;
        archive << m_Monday;
        archive << m_Tuesday;
        archive << m_Wednesday;
        archive << m_Thursday;
        archive << m_Friday;
        archive << m_Saturday;
    }
    else
    {
        archive >> m_EventType;
        archive >> m_Label;
        archive >> m_StartTime;
    }
}

```

```

        archive >> m_EndTim ;
    archive >> m_Sunday;
        archive >> m_Monday;
    archive >> m_Tuesday;
        archive >> m_Wednesday;
        archive >> m_Thursday;
        archive >> m_Friday;
        archive >> m_Saturday;
    }

    // Serialize actions
    m_Actions.Serialize ( archive );
}

CPhoneAction::CPhoneAction()
{
    m_ActionType = ACTIONTYPE_PHONE;
    m_WaitToHangUp = 20;
}

CVideoRecordAction::CVideoRecordAction()
{
    m_ActionType = ACTIONTYPE_RECORDVIDEO;
    m_UseDefaultFilename = TRUE;
    m_Duration = gm_AppConfig.m_DefaultVideoRecordDuration;
    m_KeepLeadingVideo = TRUE;
    m_Continuous = FALSE;
    m_OverwritePrior = TRUE;
}

CX10Action::CX10Action()
{
    m_ActionType = ACTIONTYPE_X10;
    m_Command = X10COMMAND_ON;
    m_DeviceCode = 0;
    m_HouseCode = 0;
    m_PercentDim = 0;
    m_Port = 0;
}

bool CEmailAction::GetActionSummary(CString& strSummary)
{
    strSummary.Format("To:%s, Subject:%s", m_To, m_Subject);
    return true;
}

bool CAudioAction::GetActionSummary(CString& strSummary)
{
    strSummary.Format("Audio File:%s", m_AudioFile);
    return true;
}

CActionsArray::~CActionsArray()
{
    // Need to manually delete pointers
    for ( int i = 0; i < GetSize(); i++ )
        delete ElementAt(i);
}

```

```

}

bool CPhoneAction::GetActionSummary(CString& strSummary)
{
    strSummary.Format("Phone:%s, Tones:%s, File:%s", m_PhoneNumber,
m_DialTones, m_AudioFile);
    return true;
}

bool CVideoRecordAction::GetActionSummary(CString& strSummary)
{
    CString t_str;

    t_str.LoadString(IDS_USEDEFAULTFILENAME);

    strSummary.Format("Duration (hh:mm:ss): %02d:%02d:%02d",
(LPCTSTR) m_Duration.GetHour(), (LPCTSTR) m_Duration.GetMinute(),
(LPCTSTR) m_Duration.GetSecond());
    return true;
}

bool CX10Action::GetActionSummary(CString& strSummary)
{
    char t_cDevice = 'A';
    CString t_strCommand;

    t_cDevice += (char) m_HouseCode;

    switch ( m_Command )
    {
    case 0:
        t_strCommand.LoadString(IDS_ON);
        break;
    case 1:
        t_strCommand.LoadString(IDS_OFF);
        break;
    case 2:
        t_strCommand.LoadString(IDS_DIM);
        break;
    }

    strSummary.Format("Turn X10 Device:%c%d %s using COM%d",
t_cDevice, m_DeviceCode+1, t_strCommand, m_Port+1 );
    return true;
}

bool CActionsArray::DeleteAction(int nActionIndex)
{
    if ( nActionIndex < 0 || nActionIndex >= GetSize() )
        return false;

    delete ElementAt(nActionIndex);
    RemoveAt(nActionIndex);

    return true;
}

```

```

bool CActionsArray::EditAction(int nActionIndex)
{
    CActionInfo * t_ActionInfo;

    if ( nActionIndex < 0 || nActionIndex >= GetSize() )
        return false;

    t_ActionInfo = ( CActionInfo * ) ElementAt(nActionIndex );

    return t_ActionInfo->EditAction();
}

void CActionsArray::DeleteAll()
{
    for (int i = 0; i < GetSize(); i++ )
        delete ElementAt(i);

    RemoveAll();
}

CEMailAction& CEMailAction::operator= ( CEMailAction& EMailAction )
{
    m_ActionType = EMailAction.m_ActionType;
    m_To = EMailAction.m_To ;
    m_Cc = EMailAction.m_Cc;
    m_Subject = EMailAction.m_Subject;
    m_Message = EMailAction.m_Message;
    m_AttachVideo = EMailAction.m_AttachVideo;
    m_Duration = EMailAction.m_Duration;
    m_KeepLeadingVideo = EMailAction.m_KeepLeadingVideo;

    return EMailAction;
}

CEMailAction::CEMailAction(CEMailAction* EMailAction)
{
    m_ActionType = EMailAction->m_ActionType;
    m_To = EMailAction->m_To ;
    m_Cc = EMailAction->m_Cc;
    m_Subject = EMailAction->m_Subject;
    m_Message = EMailAction->m_Message;
    m_AttachVideo = EMailAction->m_AttachVideo;
    m_Duration = EMailAction->m_Duration;
    m_KeepLeadingVideo = EMailAction->m_KeepLeadingVideo;
}

CAudioAction::CAudioAction(CAudioAction* AudioAction)
{
    m_ActionType = AudioAction->m_ActionType;
    m_AudioFile = AudioAction->m_AudioFile;
}

CPhoneAction::CPhoneAction(CPhoneAction * PhoneAction)
{
    m_ActionType = PhoneAction->m_ActionType;
    m_AudioFile = PhoneAction->m_AudioFile;
}

```

```

    m_DialTones = PhoneAction->m_DialTones;
    m_PhoneNumber = PhoneAction->m_PhoneNumber;
    m_WaitToHangUp = PhoneAction->m_WaitToHangUp;
}

CVideoRecordAction::CVideoRecordAction(CVideoRecordAction *
VideoRecordAction)
{
    m_ActionType = VideoRecordAction->m_ActionType;
    m_Duration = VideoRecordAction->m_Duration;
    m_Filename = VideoRecordAction->m_Filename;
    m_UseDefaultFilename = VideoRecordAction->m_UseDefaultFilename;
    m_Title = VideoRecordAction->m_Title ;
    m_KeepLeadingVideo = VideoRecordAction->m_KeepLeadingVideo;
    m_Continuous = VideoRecordAction->m_Continuous;
    m_OverwritePrior = VideoRecordAction->m_OverwritePrior;
}

CX10Action::CX10Action(CX10Action * X10Action)
{
    m_ActionType = X10Action->m_ActionType;
    m_HouseCode = X10Action->m_HouseCode;
    m_DeviceCode = X10Action->m_DeviceCode;
    m_Command = X10Action->m_Command;
    m_PercentDim = X10Action->m_PercentDim;
    m_Port = X10Action->m_Port;
}

void CActionsArray::Serialize( CArchive& archive )
{
    int t_nSize;
    int t_nActionType;

    if( archive.IsStoring() )
    {
        CActionInfo * t_ActionInfo;
        archive << GetSize();

        for ( int i = 0; i < GetSize(); i++ )
        {
            t_ActionInfo = ( CActionInfo * ) ElementAt(i);
            archive << t_ActionInfo->m_ActionType;
            t_ActionInfo = ( CActionInfo * ) ElementAt(i);
            t_ActionInfo->Serialize ( archive );
        }
    }
    else
    {
        CEmailAction * t_EmailAction;
        CPhoneAction * t_PhoneAction;
        CVideoRecordAction * t_VideoRecordAction;
        CX10Action * t_X10Action;
        CAudioAction * t_AudioAction ;

        // Clear out current array contents

```

```

DeleteAll();

// Read in new contents
archive >> t_nSize;
for ( int i = 0; i < t_nSize; i++ )
{
    archive >> t_nActionType;
    switch ( t_nActionType )
    {
        case ACTIONTYPE_EMAIL:
            t_EmailAction = new CEmailAction;
            t_EmailAction->Serialize( archive );
            Add( t_EmailAction );
            break;
        case ACTIONTYPE_PHONE:
            t_PhoneAction = new CPhoneAction;
            t_PhoneAction->Serialize( archive );
            Add( t_PhoneAction );
            break;
        case ACTIONTYPE_RECORDVIDEO:
            t_VideoRecordAction = new CVideoRecordAction;
            t_VideoRecordAction->Serialize( archive );
            Add ( t_VideoRecordAction );
            break;
        case ACTIONTYPE_X10:
            t_X10Action = new CX10Action;
            t_X10Action->Serialize( archive );
            Add ( t_X10Action );
            break;
        case ACTIONTYPE_AUDIO:
            t_AudioAction = new CAudioAction;
            t_AudioAction->Serialize( archive );
            Add ( t_AudioAction );
            break;
    }
}

}

void CEmailAction::Serialize( CArchive& archive )
{
    if( archive.IsStoring() )
    {
        archive << m_ActionType;
        archive << m_To;
        archive << m_Cc;
        archive << m_Subject;
        archive << m_Message;
        archive << m_AttachVideo;
        archive << m_Duration;
        archive << m_KeepLeadingVideo;
    }
    else
    {
        archive >> m_ActionType;
        archive >> m_To;
        archive >> m_Cc;
    }
}

```

```

        archive >> m_Subject;
        archive >> m_Message;
        archive >> m_AttachVideo;
        archive >> m_Duration;
        archive >> m_KeepLeadingVideo;
    }
}

void CAudioAction::Serialize( CArchive& archive )
{
    if( archive.IsStoring() )
    {
        archive << m_ActionType;
        archive << m_AudioFile;
    }
    else
    {
        archive >> m_ActionType;
        archive >> m_AudioFile;
    }
}

void CPhoneAction::Serialize( CArchive& archive )
{
    if( archive.IsStoring() )
    {
        archive << m_ActionType;
        archive << m_AudioFile;
        archive << m_DialTones;
        archive << m_PhoneNumber;
        archive << m_WaitToHangUp;
    }
    else
    {
        archive >> m_ActionType;
        archive >> m_AudioFile;
        archive >> m_DialTones;
        archive >> m_PhoneNumber;
        archive >> m_WaitToHangUp;
    }
}

void CVideoRecordAction::Serialize( CArchive& archive )
{
    if( archive.IsStoring() )
    {
        archive << m_ActionType;
        archive << m_Duration;
        archive << m_Filename;
        archive << m_UseDefaultFilename;
        archive << m_Title;
        archive << m_KeepLeadingVideo;
        archive << m_Continuous;
        archive << m_OverwritePrior;
    }
    else

```

```

    {
        archive >> m_ActionType;
        archive >> m_Duration;
        archive >> m_Filename;
        archive >> m_UseDefaultFilename;
        archive >> m_Title;

        archive >> m_KeepLeadingVideo;
        archive >> m_Continuous;
        archive >> m_OverwritePrior;
    }
}

void CX10Action::Serialize( CArchive& archive )
{
    if( archive.IsStoring() )
    {
        archive << m_ActionType;
        archive << m_Command;
        archive << m_DeviceCode;
        archive << m_HouseCode;
        archive << m_PercentDim;
        archive << m_Port;
    }
    else
    {
        archive >> m_ActionType;
        archive >> m_Command;
        archive >> m_DeviceCode;
        archive >> m_HouseCode;
        archive >> m_PercentDim;
        archive >> m_Port;
    }
}

bool CActionsArray::AddAction(int nActionType)
{
    bool t_bReturn = false;

    switch ( nActionType )
    {
    {
        case ACTIONTYPE_EMAIL:
            t_bReturn = AddEmailAction();
            break;
        case ACTIONTYPE_PHONE:
            t_bReturn = AddPhoneAction();
            break;
        case ACTIONTYPE_RECORDVIDEO:
            t_bReturn = AddVideoRecordAction();
            break;
        case ACTIONTYPE_X10:
            t_bReturn = AddX10Action();
            break;
        case ACTIONTYPE_AUDIO:
            t_bReturn = AddAudioAction();
            break;
    }
}

```

```

        return t_bReturn;
    }

bool CEmailAction::EditAction()
{
    bool t_bReturn = false;
    CEmailDlg t_EMailDlg;

    // Initialize dialog values
    t_EMailDlg.m_To = m_To ;
    t_EMailDlg.m_Cc = m_Cc ;
    t_EMailDlg.m_Subject = m_Subject ;
    t_EMailDlg.m_Message = m_Message ;
    t_EMailDlg.m_AttachVideo = m_AttachVideo ;
    t_EMailDlg.m_Duration = m_Duration;
    t_EMailDlg.m_KeepLeadingVideo = m_KeepLeadingVideo;

    if ( t_EMailDlg.DoModal() == IDOK )
    {
        m_To = t_EMailDlg.m_To;
        m_Cc = t_EMailDlg.m_Cc;
        m_Subject = t_EMailDlg.m_Subject;
        m_Message = t_EMailDlg.m_Message;
        m_AttachVideo = t_EMailDlg.m_AttachVideo;
        m_Duration = t_EMailDlg.m_Duration ;
        m_KeepLeadingVideo = t_EMailDlg.m_KeepLeadingVideo;

        t_bReturn = true;
    }

    return t_bReturn;
}

bool CAudioAction::EditAction()
{
    bool t_bReturn = false;
    CAudioDlg t_AudioDlg;

    // Initialize dialog values
    t_AudioDlg.m_AudioFile = m_AudioFile ;

    if ( t_AudioDlg.DoModal() == IDOK )
    {
        m_AudioFile = t_AudioDlg.m_AudioFile ;
        t_bReturn = true;
    }

    return t_bReturn;
}

bool CActionsArray::AddEmailAction()
{
    bool t_bReturn = false;
    CEmailDlg t_EMailDlg;
    CEmailAction * t_EmailAction = new CEmailAction ;

```

```

    // Initialize dialog values for a new EMail values that are
    initialized
    t_EmailDlg.m_AttachVideo = t_EmailAction->m_AttachVideo;
    t_EmailDlg.m_Duration = t_EmailAction->m_Duration ;
    t_EmailDlg.m_KeepLeadingVideo = t_EmailAction->m_KeepLeadingVideo ;
;

    if ( t_EmailDlg.DoModal() == IDOK )
    {
        t_EmailAction->m_To = t_EmailDlg.m_To;
        t_EmailAction->m_Cc = t_EmailDlg.m_Cc;
        t_EmailAction->m_Subject = t_EmailDlg.m_Subject;
        t_EmailAction->m_Message = t_EmailDlg.m_Message;
        t_EmailAction->m_AttachVideo = t_EmailDlg.m_AttachVideo;
        t_EmailAction->m_Duration = t_EmailDlg.m_Duration ;
        t_EmailAction->m_KeepLeadingVideo =
t_EmailDlg.m_KeepLeadingVideo ;

        Add(t_EmailAction);
        t_bReturn = true;
    }
    else delete t_EmailAction;

    return t_bReturn;
}

bool CPhoneAction::EditAction()
{
    bool t_bReturn = false;
    CPhoneDlg t_PhoneDlg;

    // Initialize dialog values
    t_PhoneDlg.m_PhoneNumber = m_PhoneNumber ;
    t_PhoneDlg.m_DialTones = m_DialTones ;
    t_PhoneDlg.m_AudioFile = m_AudioFile ;
    t_PhoneDlg.m_WaitToHangUp = m_WaitToHangUp ;

    if ( t_PhoneDlg.DoModal() == IDOK )
    {
        m_PhoneNumber = t_PhoneDlg.m_PhoneNumber ;
        m_DialTones = t_PhoneDlg.m_DialTones;
        m_AudioFile = t_PhoneDlg.m_AudioFile ;
        m_WaitToHangUp = t_PhoneDlg.m_WaitToHangUp ;

        t_bReturn = true;
    }

    return t_bReturn;
}

bool CActionsArray::AddPhoneAction()
{
    bool t_bReturn = false;
    CPhoneDlg t_PhoneDlg;
    CString strMessage;

    // If no TAPI device is selected then warn user and give

```

```

// the option to cancel
if ( gm_AppConfig.m_TAPIDevice.IsEmpty() )
{
    strMessage.LoadString(IDS_ERR_NOTAPIDEVICESELECTED);
    if ( MessageBox(GetFocus(), strMessage, NULL, MB_OKCANCEL)
== IDCANCEL )
        return false;
}

CPhoneAction * t_PhoneAction = new CPhoneAction ;

t_PhoneDlg.m_WaitToHangUp = t_PhoneAction->m_WaitToHangUp ;

if ( t_PhoneDlg.DoModal() == IDOK )
{
    t_PhoneAction->m_PhoneNumber = t_PhoneDlg.m_PhoneNumber ;
    t_PhoneAction->m_DialTones = t_PhoneDlg.m_DialTones;
    t_PhoneAction->m_AudioFile = t_PhoneDlg.m_AudioFile ;
    t_PhoneAction->m_WaitToHangUp = t_PhoneDlg.m_WaitToHangUp;

    Add(t_PhoneAction);
    t_bReturn = true;
}
else delete t_PhoneAction;

return t_bReturn;
}

bool CActionsArray::AddAudioAction()
{
    bool t_bReturn = false;
    CAudioDlg t_AudioDlg;
    CAudioAction * t_AudioAction = new CAudioAction ;

    if ( t_AudioDlg.DoModal() == IDOK )
    {
        t_AudioAction->m_AudioFile = t_AudioDlg.m_AudioFile;

        Add(t_AudioAction);
        t_bReturn = true;
    }
    else delete t_AudioAction;

    return t_bReturn;
}

bool CX10Action::EditAction()
{
    bool t_bReturn = false;
    CX10Dlg t_X10Dlg;

    // Initialize dialog values
    t_X10Dlg.m_HouseCode = m_HouseCode ;
    t_X10Dlg.m_DeviceCode = m_DeviceCode ;
    t_X10Dlg.m_Command = m_Command ;
    t_X10Dlg.m_PercentDim = m_PercentDim ;
    t_X10Dlg.m_Port = m_Port ;

```

```

if ( t_X10Dlg.DoModal() == IDOK )
{
    m_HouseCode = t_X10Dlg.m_HouseCode ;
    m_DeviceCode = t_X10Dlg.m_DeviceCode ;
    m_Command = t_X10Dlg.m_Command ;
    m_PercentDim = t_X10Dlg.m_PercentDim ;
    m_Port = t_X10Dlg.m_Port ;

    t_bReturn = true;
}

return t_bReturn;
}

bool CActionsArray::AddX10Action()
{
    bool t_bReturn = false;
    CX10Dlg t_X10Dlg;
    CX10Action * t_X10Action = new CX10Action ;

    // Initialize dialog values for a new X10 values that are
    initialized
    t_X10Dlg.m_HouseCode = t_X10Action->m_HouseCode ;
    t_X10Dlg.m_DeviceCode = t_X10Action->m_DeviceCode ;
    t_X10Dlg.m_Command = t_X10Action->m_Command ;
    t_X10Dlg.m_PercentDim = t_X10Action->m_PercentDim ;
    t_X10Dlg.m_Port = t_X10Action->m_Port ;

    if ( t_X10Dlg.DoModal() == IDOK )
    {
        t_X10Action->m_HouseCode = t_X10Dlg.m_HouseCode ;
        t_X10Action->m_DeviceCode = t_X10Dlg.m_DeviceCode ;
        t_X10Action->m_Command = t_X10Dlg.m_Command ;
        t_X10Action->m_PercentDim = t_X10Dlg.m_PercentDim ;
        t_X10Action->m_Port = t_X10Dlg.m_Port ;

        Add(t_X10Action);
        t_bReturn = true;
    }
    else delete t_X10Action;

    return t_bReturn;
}

bool CVideoRecordAction::EditAction()
{
    bool t_bReturn = false;
    CVideoRecordDlg t_VideoRecordDlg;

    // Initialize dialog values for a new Video Record values that
    are initialized
    t_VideoRecordDlg.m_Duration = m_Duration ;
    t_VideoRecordDlg.m_UseDefaultFilename = m_UseDefaultFilename ;
    t_VideoRecordDlg.m_Filename = m_Filename ;
    t_VideoRecordDlg.m_Title = m_Title ;

```

```

t_VideoRecordDlg.m_KeepLeadingVideo = m_KeepLeadingVideo;
t_VideoRecordDlg.m_Continuous = m_Continuous ;
t_VideoRecordDlg.m_OverwritePrior = m_OverwritePrior ;

if ( t_VideoRecordDlg.DoModal() == IDOK )
{
    m_Duration = t_VideoRecordDlg.m_Duration ;
    m_Filename = t_VideoRecordDlg.m_Filename ;
    m_UseDefaultFilename =
t_VideoRecordDlg.m_UseDefaultFilename ;
    m_Title = t_VideoRecordDlg.m_Title ;
    m_KeepLeadingVideo = t_VideoRecordDlg.m_KeepLeadingVideo ;
    m_Continuous = t_VideoRecordDlg.m_Continuous ;
    m_OverwritePrior = t_VideoRecordDlg.m_OverwritePrior ;

    t_bReturn = true;
}

return t_bReturn;
}

bool CActionsArray::AddVideoRecordAction()
{
    bool t_bReturn = false;
    CVideoRecordDlg t_VideoRecordDlg;
    CVideoRecordAction * t_VideoRecordAction = new CVideoRecordAction
;

    // Only one video record action for each event
    for (int i = 0; i < GetSize(); i++)
    {
        CActionInfo* t_ActionInfo = (CActionInfo*) ElementAt(i);
        if ( t_ActionInfo->m_ActionType == ACTIONTYPE_RECORDVIDEO )
        {
            CString t_strMessage;

            t_strMessage.LoadString(IDS_ERR_ONLYONEVIDEORECORDACTION);
            MessageBox(NULL, t_strMessage, NULL, MB_OK);
            delete t_VideoRecordAction;
            goto Exit1;
        }
    }

    // Initialize dialog values for a new Video Record values that
    are initialized
    t_VideoRecordDlg.m_Duration = t_VideoRecordAction->m_Duration ;
    t_VideoRecordDlg.m_UseDefaultFilename = t_VideoRecordAction-
>m_UseDefaultFilename ;
    t_VideoRecordDlg.m_KeepLeadingVideo = t_VideoRecordAction-
>m_KeepLeadingVideo ;
    t_VideoRecordDlg.m_Continuous = t_VideoRecordAction->m_Continuous
;
    t_VideoRecordDlg.m_OverwritePrior = t_VideoRecordAction-
>m_OverwritePrior ;

    if ( t_VideoRecordDlg.DoModal() == IDOK )
    {

```

```

        t_VideoRecordAction->m_Duration =
t_VideoRecordDlg.m_Duration ;
        t_VideoRecordAction->m_Filename =
t_VideoRecordDlg.m_Filename ;
        t_VideoRecordAction->m_UseDefaultFilename =
t_VideoRecordDlg.m_UseDefaultFilename ;
        t_VideoRecordAction->m_Title = t_VideoRecordDlg.m_Title ;
        t_VideoRecordAction->m_KeepLeadingVideo =
t_VideoRecordDlg.m_KeepLeadingVideo ;
        t_VideoRecordAction->m_Continuous =
t_VideoRecordDlg.m_Continuous ;
        t_VideoRecordAction->m_OverwritePrior =
t_VideoRecordDlg.m_OverwritePrior ;

        Add(t_VideoRecordAction);
        t_bReturn = true;
    }
    else delete t_VideoRecordAction;

Exit1:
    return t_bReturn;
}

CConnectionMsg::CConnectionMsg()
: CObject()
{
    m_nSize = sizeof(CConnectionMsg);
    lstrcpy(m_szMessage, "");
    lstrcpy(m_szLabel, "");
    m_nMessageType = CONNECTIONMSGTYPE_IMMEDIATE;
    m_bLocal = true;
}

bool CConnectionMsg::SetLabel(CString strLabel)
{
    lstrcpy(m_szLabel, strLabel, 64);
    return true;
}

bool CConnectionMsg::SetMessage(CString strMessage)
{
    lstrcpy(m_szMessage, strMessage, 128);
    return true;
}

UINT ExecuteEmailActionThread(LPVOID lParam)
{
    CEmailAction* t_EmailAction;
    CString t_strSummary;
    CSimpleMAPI t_SimpleMAPI;

    EnterCriticalSection(&gm_csEmailAction);

    t_EmailAction = (CEmailAction*) lParam;
    t_strSummary.Format("To:%s, Subject:%s, Attach Video: %s",
t_EmailAction->m_To, t_EmailAction->m_Subject);

```

```

        t_SimpleMAPI.QuickSendMail(t_EMailAction->m_To, t_EMailAction-
>m_Cc, t_EMailAction->m_Subject, t_EMailAction->m_Message, "");

        LeaveCriticalSection(&gm_csEMailAction);

        return 0;
    }

bool CEMailAction::ExecuteAction(CConnectionInfo* pConnectionInfo)
{
    bool t_bReturn = true;
    CSimpleMAPI t_SimpleMAPI;
    CString t_strVideoFile;
/*
    if ( m_AttachVideo )
    {
        CVideoRecordAction t_VideoRecordAction;

        t_VideoRecordAction.m_Title = pConnectionInfo->m_Label;
        t_VideoRecordAction.m_Duration = m_Duration;
        t_VideoRecordAction.m_KeepLeadingVideo = m_KeepLeadingVideo
;

        if ( t_VideoRecordAction.Trigger(pConnectionInfo) )
            GetMostRecentVideoFile(t_strVideoFile);
    }
*/

    // t_bReturn = t_SimpleMAPI.QuickSendMail(m_To, m_Cc, m_Subject,
m_Message, t_strVideoFile);

    AfxBeginThread(ExecuteEMailActionThread, (LPVOID) this);

    return t_bReturn;
}

UINT ExecuteAudioActionThread(LPVOID lParam)
{
    CSimpleDirectAudio t_SimpleDirectAudio;
    CAudioAction* t_AudioAction;

    HRESULT t_hResult = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED
);
    if ( FAILED( t_hResult ) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGE_TYPE_ERROR,
"CoInitializeEx", t_hResult );
        return 1;
    }

    EnterCriticalSection(&gm_csAudioAction);
    t_AudioAction = (CAudioAction* ) lParam;

    t_SimpleDirectAudio.QuickPlayAudioFile(t_AudioAction-
>m_AudioFile);
    LeaveCriticalSection(&gm_csAudioAction);

```

```

        CoUninitialize();

        r turn 0;
    }

bool CAudioAction::ExecuteAction()
{
    //    bool t_bReturn = false;
    //    CSimpleDirectAudio t_SimpleDirectAudio;
    //    t_bReturn = t_SimpleDirectAudio.QuickPlayAudioFile(m_AudioFile);

    bool t_bReturn = true;
    AfxBeginThread(ExecuteAudioActionThread, (LPVOID) this);
    return t_bReturn;
}

#define PROJNAL_CONNECTIONFILETIME CTime(2002, 5, 15, 10, 30, 0)

bool CConnectionsArray::LoadSettings()
{
    CString t_strDefaultFilename;
    CFile t_File;
    CFileStatus t_FileStatus;

    GetSerializeFileName(t_strDefaultFilename);
    if ( !t_File.Open(t_strDefaultFilename, CFile::modeRead) )
        return false;

    // Make sure we are not reading an old file
    t_File.GetStatus(t_FileStatus);
    if ( t_FileStatus.m_mtime > PROJNAL_CONNECTIONFILETIME )
    {
        CArchive t_archive(&t_File, CArchive::load);
        Serialize( t_archive );
        t_archive.Close();
    }
    t_File.Close();

    return true;
}

void CConnectionsArray::SaveSettings()
{
    CString strFileName;
    CFile t_File;

    GetSerializeFileName(strFileName);

    if ( !t_File.Open((LPCTSTR) strFileName, CFile::modeWrite |
CFile::modeCreate ) )
        return;

    CArchive t_archive(&t_File, CArchive::store);

    Serialize( t_archive );

```

```

        t_archive.Close();
        t_File.Close();
    }

bool CConnectionsArray::GetSerializeFileName(CString& strFileName)
{
    CString t_strDefaultFilename;
    char t_szSavePath[_MAX_PATH];

    lstrcpy (t_szSavePath, gm_AppConfig.m_StartupPath);

    // Create save path
    t_strDefaultFilename.LoadString(IDS_DEFAULTPROFILEFILENAME);
    PathAppend(t_szSavePath, t_strDefaultFilename);

    strFileName = t_szSavePath;
    return true;
}

UINT ExecuteX10ActionThread(LPVOID lParam)
{
    CX10 m_X10;
    short t_Command;
    CX10Action* t_X10Action;

    HRESULT t_hResult = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED
);
    if ( FAILED( t_hResult ) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGE_TYPE_ERROR,
"ExecuteX10ActionThread", t_hResult );
        return 1;
    }

    EnterCriticalSection(&gm_csX10Action);
    t_X10Action = (CX10Action* ) lParam;

    switch ( t_X10Action->m_Command )
    {
    case 0: // On
        t_Command = CM17ACOMMAND_ON;
        break;
    case 1: // Off
        t_Command = CM17ACOMMAND_OFF;
        break;
    case 2: // Dim
        t_Command = CM17ACOMMAND_DIM;
        break;
    }

    // Set port first since it is activated when X10 is initialized
    // Port starts at 1
    m_X10.SetPort(t_X10Action->m_Port + 1);

    if ( !m_X10.Initialize() )
        goto Exit1;
}

```

```

        m_X10.ExecuteCommand((short) t_X10Action->m_HouseCode, (short)
t_X10Action->m_DeviceCode + 1, t_Command, (short) t_X10Action-
>m_PercentDim);

        m_X10.Uninitialize();
Exit1:
        LeaveCriticalSection(&gm_csX10Action);
        CoUninitialize();

        return 0;
}

bool CX10Action::ExecuteAction()
{
    /*
        bool t_bReturn = false;
        CX10 m_X10;
        short t_Command;

        switch ( m_Command )
        {
        case 0: // On
            t_Command = CM17ACOMMAND_ON;
            break;
        case 1: // Off
            t_Command = CM17ACOMMAND_OFF;
            break;
        case 2: // Dim
            t_Command = CM17ACOMMAND_DIM;
            break;
        }

        // Set port first since it is activated when X10 is initialized
        // Port starts at 1
        m_X10.SetPort(m_Port + 1);

        if ( !m_X10.Initialize() )
            goto Exit1;

        m_X10.ExecuteCommand((short) m_HouseCode, (short) m_DeviceCode +
1, t_Command, (short) m_PercentDim);

        m_X10.Uninitialize();
        t_bReturn = true;
Exit1:
        return t_bReturn;
    */

    AfxBeginThread(ExecuteX10ActionThread, (LPVOID) this);
    return true;
}

UINT ExecutePhoneActionThread(LPVOID lParam)
{
    CTAPIControl t_TAPIControl;
    CPhoneAction* t_PhoneAction;

```

```

        HRESULT t_hResult = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED
    );
        if ( FAILED( t_hResult ) )
        {
            NSENDFEEDBACKHRESULT(FEEDBACKMESSAGE_TYPE_ERROR,
"ExecutePhoneActionThread", t_hResult );
            return 1;
        }

        EnterCriticalSection(&gm_csPhoneAction);
        t_PhoneAction = (CPhoneAction* ) lParam;
        t_TAPIControl.QuickCall(gm_AppConfig.m_TAPIDevice, t_PhoneAction-
>m_PhoneNumber, t_PhoneAction->m_DialTones, t_PhoneAction->m_AudioFile,
t_PhoneAction->m_WaitToHangUp );

        LeaveCriticalSection(&gm_csPhoneAction);

        CoUninitialize();

        return 0;
    }

bool CPhoneAction::ExecuteAction()
{
    // CTAPIControl t_TAPIControl;
    // return t_TAPIControl.QuickCall(gm_AppConfig.m_TAPIDevice,
m_PhoneNumber, m_DialTones, m_AudioFile, m_WaitToHangUp );

    AfxBeginThread(ExecutePhoneActionThread, (LPVOID) this);
    return true;
}

bool CVideoRecordAction::ExecuteAction(CConnectionInfo*
pConnectionInfo, CEventInfo* pEventInfo)
{
    int t_RecordTrigger;

    if ( pEventInfo->m_EventType == EVENTTYPE_ALARM )
        t_RecordTrigger = RECORDTRIGGER_ALARM;
    else
        t_RecordTrigger = RECORDTRIGGER_SCHEDULEDEVENT;

    if ( m_Continuous )
        t_RecordTrigger += 1;

    pConnectionInfo->m_SimpleVideo.SetRecordTrigger(t_RecordTrigger
);
    m_Title = pConnectionInfo->m_Label;
    pConnectionInfo->m_SimpleVideo.RecordFromTrigger(this);
    return true;
}

//-----
// File: DirectPlay.cpp
//

```

```

// Desc: DirectPlay-related classes for managing individual and groups
of connections filter for detecting motion in a video stream
//
// Comments:
//
// Debug Notes: Since debugging DirectPlay is very cumbersome, a set of
debug outputs has been coded.
//               They now use the macro NUPDATE... to simplify
the coding
//
//
// History: 03/13/02      LCK          Created
//
//
// Copyright (c) 2002 BKLK Inc.  All rights reserved.
//-----
-----

#include "stdafx.h"
#include "Project Nalay.h"

#include "objbase.h"
// #include "initguid.h"
#include "dplay8guids.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"
#include "ConnectionDlg.h"
#include "EventDialog.h"

IMPLEMENT_SERIAL( CConnectionInfo, CObject, 1 )

extern CRITICAL_SECTION gm_csHostList;

//-----
// Miscellaneous helper functions
//-----
#define SAFE_DELETE(p)          { if(p) { delete (p);      (p)=NULL; } }
#define SAFE_DELETE_ARRAY(p)   { if(p) { delete[] (p);    (p)=NULL; } }
#define SAFE_RELEASE(p)        { if(p) { (p)->Release(); (p)=NULL; } }

//-----
// Function-prototypes
//-----
HRESULT WINAPI DirectPlayMessageHandler(PVOID pvUserContext, DWORD
dwMessageId, PVOID pMsgBuffer);

//-----
// Name: DirectPlayMessageHandler
// Desc: Handler for DirectPlay messages.  This tutorial doesn't repond
to any

```

```

//      DirectPlay messages
//-----
static TCHAR DPMbuffer[] = "DirectPlayMessageHandler";
HRESULT WINAPI DirectPlayMessageHandler( PVOID pvUserContext, DWORD
dwMessageId,
                                     PVOID pMsgBuffer)
{
    HRESULT t_hrResult = S_OK;

    EnterCriticalSection(&gm_csHostList);
    switch( dwMessageId )
    {
        case DPN_MSGID_RECEIVE:
            {
                PDPNMSG_RECEIVE t_pMsg;
                CConnectionMsg* t_pConnectionMsg;

                t_pMsg = (PDPNMSG_RECEIVE) pMsgBuffer;
                t_pConnectionMsg = (CConnectionMsg*) t_pMsg-
>pReceiveData;

                switch ( t_pConnectionMsg->m_nMessageType )
                {
                    case CONNECTIONMSGTYPE_IMMESSAGE:
                        SendIMMessage(t_pConnectionMsg);
                        NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_RECEIVE", t_pConnectionMsg->m_szLabel);
                        break;
                    case CONNECTIONMSGTYPE_LOCALVIDEOEVENTS:
                        SendIMMessage(t_pConnectionMsg);
                        NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_RECEIVE2", t_pConnectionMsg->m_szLabel);
                        NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_RECEIVE2", t_pConnectionMsg->m_szMessage);
                        break;
                }
            }
            break;
        case DPN_MSGID_ADD_PLAYER_TO_GROUP:
            //      NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_ADD_PLAYER_TO_GROUP", "");
            break;
        case DPN_MSGID_APPLICATION_DESC:
            //      NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_APPLICATION_DESC", "");
            break;
        case DPN_MSGID_ASYNC_OP_COMPLETE:
            //      NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_ASYNC_OP_COMPLETE", "");
            break;
        case DPN_MSGID_CLIENT_INFO:
            //      NSENDMESSAGE(FEEDBACKMSGTYPE_DEBUG,
"DPN_MSGID_CLIENT_INFO", "");
            break;
        case DPN_MSGID_CONNECT_COMPLETE:
            {

```

```

        CString t_str;
        PDPNMSG_CONNECT_COMPLETE t_pMsgConnectComplete;
        t_pMsgConnectComplete = (PDPNMSG_CONNECT_COMPLETE)
pMsgBuffer;
        t_str.Format("HRESULT = %ld", t_pMsgConnectComplete-
>hResultCode);
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_CONNECT_COMPLETE", t_str);
    }
    break;
    case DPN_MSGID_CREATE_GROUP:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_CREATE_GROUP", "");
    break;
    case DPN_MSGID_CREATE_PLAYER:
    {
        CString t_str;
        PDPNMSG_CREATE_PLAYER t_pMsgCreatePlayer;
        t_pMsgCreatePlayer = (PDPNMSG_CREATE_PLAYER ) pMsgBuffer;
        t_str.Format("DPNID = %ld, Player = %ld",
t_pMsgCreatePlayer->dpnidPlayer, (long) t_pMsgCreatePlayer-
>pvPlayerContext);
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_CREATE_PLAYER", t_str);
    }
    break;
    case DPN_MSGID_DESTROY_GROUP:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_DESTROY_GROUP", "");
    break;
    case DPN_MSGID_DESTROY_PLAYER:
    {
        CString t_str;
        PDPNMSG_DESTROY_PLAYER t_pMsgDestroyPlayer;
        t_pMsgDestroyPlayer = (PDPNMSG_DESTROY_PLAYER ) pMsgBuffer;
        t_str.Format("DPNID = %ld, Reason: %ld",
t_pMsgDestroyPlayer->dpnidPlayer, t_pMsgDestroyPlayer->dwReason);
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_DESTROY_PLAYER", t_str);
    }
    break;
    case DPN_MSGID_ENUM_HOSTS_QUERY:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_ENUM_HOSTS_QUERY", "");
    break;
    case DPN_MSGID_ENUM_HOSTS_RESPONSE:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_ENUM_HOSTS_RESPONSE", "");
    break;
    case DPN_MSGID_GROUP_INFO:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_GROUP_INFO", "");
    break;
    case DPN_MSGID_HOST_MIGRATE:
        //      NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
        "DPN_MSGID_HOST_MIGRATE", "");
    break;

```

```

        case DPN_MSGID_INDICATE_CONNECT:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_INDICATE_CONNECT", "");
            break;
        case DPN_MSGID_INDICATED_CONNECT_ABORTED:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_INDICATED_CONNECT_ABORTED", "");
            break;
        case DPN_MSGID_PEER_INFO:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_PEER_INFO", "");
            break;
        case DPN_MSGID_REMOVE_PLAYER_FROM_GROUP:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_REMOVE_PLAYER_FROM_GROUP", "");
            break;
        case DPN_MSGID_RETURN_BUFFER:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_RETURN_BUFFER", "");
            break;
        case DPN_MSGID_SEND_COMPLETE:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_SEND_COMPLETE", "");
            break;
        case DPN_MSGID_SERVER_INFO:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_SERVER_INFO", "");
            break;
        case DPN_MSGID_TERMINATE_SESSION:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_TERMINATE_SESSION", "");
            break;
        default:
//          NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGETYPE_DEBUG,
"DPN_MSGID_???", "DirectPlayMessageHandler");
            break;
    }
    LeaveCriticalSection(&gm_csHostList);

    return t_hrResult ;
}

CConnectionInfo::CConnectionInfo()
{
    Initialize();
}

CConnectionInfo::~CConnectionInfo()
{
    Disconnect();
}

void CConnectionInfo::Initialize()
{
    m_ConnectionMethod = CONNECTION_LOCALTCPIP;
    m_ConnectionType = CONTYPE_VIDEOSURVEILLANCE;
}

```

```

CConnectionInfo::CConnectionInfo(CConnectionInfo& ConnectionInfo)
{
    SetCopy(ConnectionInfo);
}

void CConnectionInfo::GetCopy(CConnectionInfo& ConnectionInfo)
{
    ConnectionInfo.m_ConnectionMethod = m_ConnectionMethod ;
    ConnectionInfo.m_ConnectionType = m_ConnectionType ;
    ConnectionInfo.m_DialUpNumber = m_DialUpNumber ;
    ConnectionInfo.m_IPAddress = m_IPAddress ;
    ConnectionInfo.m_Label = m_Label ;
    ConnectionInfo.m_Password = m_Password ;
    ConnectionInfo.m_Username = m_Username ;
    ConnectionInfo.m_YellowPagesEntry = m_YellowPagesEntry ;

    for (int i = 0; i < m_Events.GetSize(); i++ )
        ConnectionInfo.m_Events.Add(m_Events.ElementAt(i) );

    ConnectionInfo.m_SimpleVideo = m_SimpleVideo;
}

CConnectionInfo& CConnectionInfo::operator = (CConnectionInfo&
ConnectionInfo)
{
    SetCopy(ConnectionInfo);
    return ConnectionInfo;
}

void CConnectionInfo::SetCopy(CConnectionInfo& ConnectionInfo)
{
    m_ConnectionMethod = ConnectionInfo.m_ConnectionMethod ;
    m_ConnectionType = ConnectionInfo.m_ConnectionType ;
    m_DialUpNumber = ConnectionInfo.m_DialUpNumber ;
    m_IPAddress = ConnectionInfo.m_IPAddress ;
    m_Label = ConnectionInfo.m_Label ;
    m_Password = ConnectionInfo.m_Password ;
    m_Username = ConnectionInfo.m_Username ;
    m_YellowPagesEntry = ConnectionInfo.m_YellowPagesEntry ;

    // First clear out events - then copy new ones in
    m_Events.RemoveAll();

    for (int i = 0; i < ConnectionInfo.m_Events.GetSize(); i++ )
        m_Events.Add(ConnectionInfo.m_Events.ElementAt(i) );

    m_SimpleVideo = ConnectionInfo.m_SimpleVideo ;
}

bool CConnectionInfo::GetConnectionMethodString(CString&
ConnectionMethod)
{
    bool t_bReturn = true;

    // Clear out existing content
    ConnectionMethod.Empty();

```

```

// Fill in with correct info
switch (m_ConnectionMethod)
{
case CONNECTION_IPADDRESS:
    ConnectionMethod.LoadString(IDS_IPADDRESS);
    break;
case CONNECTION_YELLOWPAGES:
    ConnectionMethod.LoadString(IDS_YELLOWPAGES);
    break;
case CONNECTION_LOCALTCPIP:
    ConnectionMethod.LoadString(IDS_LOCALTCPIP);
    break;
default:
    break;
}

return t_bReturn;
}

#include "atlbase.h"
#import "..\Yellow Pages\VPD.ocx" named_guids no_namespace
bool CConnectionInfo::GetIPAddress(CString& strIPAddress)
{
    bool t_bReturn = true;

    if ( m_ConnectionMethod == CONNECTION_YELLOWPAGES )
    {
        CComPtr<IVideoPeer> t_pYellowPages;
        CComBSTR t_bstrLabel = m_YellowPagesEntry;
        CComBSTR t_bstrIPAddress;
        HRESULT t_hResult;
        _bstr_t t_bstr;

        t_hResult =
t_pYellowPages.CoCreateInstance(CLSID_VideoPeer);
        if ( FAILED(t_hResult) )
        {
            NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
"GetIPAddress", t_hResult);
            return false;
        }

        t_hResult = t_pYellowPages->raw_Lookup(lptNone,
&t_bstrLabel.m_str, &t_bstrIPAddress.m_str);
        if ( FAILED(t_hResult) )
        {
            NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
"GetIPAddress", t_hResult);
            return false;
        }

        t_pYellowPages.Release();

        t_bstr = t_bstrIPAddress;
        m_IPAddress = (char*) t_bstr ;
    }
}

```

```

        strIPAddr ss = m_IPAddress;

        return t_bReturn;
    }

bool CConnectionInfo::GetConnectionKeyData(CString& KeyData )
{
    bool t_bReturn = true;

    // Clear out existing content
    KeyData.Empty();

    // Fill in with correct info
    switch (m_ConnectionMethod)
    {
        case CONNECTION_IPADDRESS:
            KeyData = m_IPAddress;
            break;
        case CONNECTION_YELLOWPAGES:
            KeyData = m_YellowPagesEntry;
            break;
        case CONNECTION_LOCALTCPIP:
        default:
            break;
    }

    return t_bReturn;
}

bool CConnectionInfo::GetConnectionTypeString(CString& ConnectionType)
{
    bool t_bReturn = true;

    // Clear out existing content
    ConnectionType.Empty();

    // Fill in with correct info
    switch (m_ConnectionType)
    {
        case CONTYPE_VIDEOSURVEILLANCE:
            ConnectionType.LoadString(IDS_VIDEOSURVEILLANCE);
            break;
        case CONTYPE_INSTANTMESSENGER:
            ConnectionType.LoadString(IDS_INSTANTMESSENGER);
            break;
        case CONTYPE_VIDEOCONFERENCE:
        default:
            ConnectionType.LoadString(IDS_VIDEOCONFERENCE);
            break;
    }

    return t_bReturn;
}

bool CConnectionInfo::Connect(HWND hWnd)
{

```

```

bool t_bReturn = false;

m_SimpleDirectPlay.m_strSessionName = m_Label;
m_SimpleDirectPlay.m_strIPAddress = m_IPAddress;
m_SimpleDirectPlay.m_strPassword = m_Password;

// depending on whether it is local or remote
switch( m_ConnectionMethod )
{
case CONNECTION_LOCALTCPIP:
//      m_SimpleVideo.Connect(hWnd, m_Label);

      if ( m_SimpleVideo.Connect(hWnd, m_Label) )
          if ( m_SimpleDirectPlay.ConnectHostSession() )
              t_bReturn = true;
          else
              m_SimpleVideo.Disconnect();

      break;

case CONNECTION_IPADDRESS:
case CONNECTION_YELLOWPAGES:
      t_bReturn = m_SimpleDirectPlay.ConnectRemoteSession();
      break;

default:
      NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_DEBUG, "Connect",
E_INVALIDARG);
      break;
}

return t_bReturn;
}

bool CConnectionInfo::Disconnect()
{
    bool t_bReturn = false;

    // depending on whether it is local or remote
    switch( m_ConnectionMethod )
    {
    case CONNECTION_LOCALTCPIP:
        m_SimpleVideo.Disconnect();
        t_bReturn = m_SimpleDirectPlay.DisconnectHostSession();
        break;

    case CONNECTION_IPADDRESS:
    case CONNECTION_YELLOWPAGES:
        t_bReturn = m_SimpleDirectPlay.DisconnectRemoteSession();
        break;

    default:
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_DEBUG, "Connect",
E_INVALIDARG);
        break;
    }
}

```

```

        return t_bReturn;
    }

void CConnectionInfo::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
    {
        archive << m_Label;
        archive << m_IPAddress;
        archive << m_YellowPagesEntry;
        archive << m_DialUpNumber;
        archive << m_ConnectionMethod;
        archive << m_ConnectionType;
        archive << m_Username;
        archive << m_Password;
    }
    else
    {
        archive >> m_Label;
        archive >> m_IPAddress;
        archive >> m_YellowPagesEntry;
        archive >> m_DialUpNumber;
        archive >> m_ConnectionMethod;
        archive >> m_ConnectionType;
        archive >> m_Username;
        archive >> m_Password;
    }

    // Serialize the events
    m_Events.Serialize( archive );

    // Seriealize Video info
    m_SimpleVideo.Serialize ( archive );
}

bool CConnectionInfo::IsEventLabelValid(CString strEventLabel)
{
    return m_Events.IsLabelValid(strEventLabel);
}

bool CConnectionInfo::AddEvent(int nEventType)
{
    bool t_bReturn = false;
    CEventDialog t_EventDialog;
    CEventInfo t_EventInfo;

    t_EventDialog.m_EventInfo.m_EventType = nEventType;
    t_EventDialog.m_ConnectionLabel = m_Label;

    if ( t_EventDialog.DoModal() == IDOK )
    {
        t_EventInfo = t_EventDialog.m_EventInfo;
    }
}

```

```

        m_Events.Add(t_EventInfo);
        t_bReturn = true;
    }

    return t_bReturn;
}

bool CConnectionInfo::TriggerScheduledEvent(CTime t_CurrentTime)
{
    // Only relevant if activated
    // LCK Add Code

    // Only relevant to video surveillance
    if ( m_ConnectionType != CONTYPE_VIDEOSURVEILLANCE )
        return false;

    // Only relevant to local connections
    if ( m_ConnectionMethod != CONNECTION_LOCALTCPIP )
        return false;

    for ( int i = 0; i < m_Events.GetSize(); i++ )
        m_Events.ElementAt(i).TriggerScheduledEvent(t_CurrentTime,
this);

    return true;
}

bool CConnectionInfo::TriggerAlarm()
{
    // Only relevant to video surveillance
    if ( m_ConnectionType != CONTYPE_VIDEOSURVEILLANCE )
        return false;

    // Only relevant to local connections
    if ( m_ConnectionMethod != CONNECTION_LOCALTCPIP )
        return false;

    for ( int i = 0; i < m_Events.GetSize(); i++ )
        m_Events.ElementAt(i).TriggerAlarm(this);

    return true;
}

bool CConnectionInfo::IsConnectionEstablished()
{
    if ( m_SimpleDirectPlay.m_State == SIMPLEDIRECTPLAYSTATE_ACTIVE )
        return true;
    return false;
}

bool CConnectionInfo::SendEvents()
{
    bool t_bReturn = false;
    CConnectionMsg t_ConnectionMsg;
    CString t_strDefaultFilename;
    CMemFile t_MemFile;

```

```

    DWORD t_dwMemSize;
    BYTE* t_pData;

    // Establish archive into memory file
    CArchive t_archive(&t_MemFile, CArchive::store);

    if ( !IsConnectionEstablished() )
    {
        NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_DEBUG, m_Label,
IDS_ERR_CONNECTIONNOTESTABLISHED);
        goto Exit1;
    }

    // Serialize data into memory
    m_Events.Serialize( t_archive );

    // Need to flush data since it is typically a small amount
    t_archive.Flush();
    t_MemFile.Flush();
    t_dwMemSize = t_MemFile.GetLength();

    // Copy events to the CConnectionMsg structure
    t_pData = t_MemFile.Detach();
    memcpy(t_ConnectionMsg.m_pData, t_pData, t_dwMemSize);
    free(t_pData);

    // Setup message handler
    t_ConnectionMsg.m_nMessageType =
CONNECTIONMESGTYPE_LOCALVIDEOREVENTS;
    t_ConnectionMsg.SetLabel(m_Label);
    t_ConnectionMsg.m_bLocal = true;

    // Send message to other connections
    t_bReturn = m_SimpleDirectPlay.SendMessage(t_ConnectionMsg);

Exit1:
    return t_bReturn;
}

bool CConnectionInfo::ReceiveEvents(CConnectionMsg* pConnectionMsg)
{
    bool t_bReturn = false;
    CMemFile t_MemFile;
    // CEventsArray t_Events;
    BYTE t_pData[CONNECTIONMESG_BUFFERLENGTH];

    // Establish memory file and its associated archive
    memcpy(t_pData, pConnectionMsg->m_pData,
CONNECTIONMESG_BUFFERLENGTH);
    t_MemFile.Attach(&t_pData[0], CONNECTIONMESG_BUFFERLENGTH);
    CArchive t_archive(&t_MemFile, CArchive::load);

    if ( !IsConnectionEstablished() )
    {
        NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_DEBUG, m_Label,
IDS_ERR_CONNECTIONNOTESTABLISHED);
        goto Exit1;
    }

```

```

    }

    // Flush data - generally a small amount
    t_archive.Flush();
    t_MemFile.Flush();

    // Serialize from memory
    m_Events.Serialize( t_archive);

    // Re flush just in case it does not bother to write data to
memory
    t_archive.Flush();
    t_MemFile.Flush();

    // Clean up memory file
    t_MemFile.Detach();
    t_MemFile.Close();
/*
    gm_Connections[t_nIndex].m_Events.RemoveAll();
    for ( i = 0; i < t_Events.GetSize() ; i++ )
    {
        CEventInfo& t_EventInfo = t_Events[i];
        gm_Connections[t_nIndex].m_Events.Add(t_EventInfo );
    }
*/
    t_bReturn = true;
Exit1:
    return t_bReturn;
}
// EMailDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Naylay.h"
#include "EMailDlg.h"
#include "SimpleMAPI.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
/////
// CEMailDlg dialog

CEMailDlg::CEMailDlg(CWnd* pParent /*=NULL*/)
: CDialog(CEMailDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CEMailDlg)
    m_AttachVideo = FALSE;
    m_Message = _T("");
    m_Subject = _T("");
    m_To = _T("");
    m_Cc = _T("");

```

```

    m_Duration = 0;
    m_KeepLeadingVideo = FALSE;
    //}}AFX_DATA_INIT
}

void CEMailDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CEMailDlg)
    DDX_Control(pDX, IDC_DURATION, m_DurationCtrl);
    DDX_Check(pDX, IDC_ATTACHVIDEO, m_AttachVideo);
    DDX_Text(pDX, IDC_MESSAGE, m_Message);
    DDX_Text(pDX, IDC_SUBJECT, m_Subject);
    DDX_Text(pDX, IDC_TO, m_To);
    DDX_Text(pDX, IDC_CC, m_Cc);
    DDX_DateTimeCtrl(pDX, IDC_DURATION, m_Duration);
    DDX_Check(pDX, IDC_KEEPLADINGVIDEO, m_KeepLeadingVideo);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CEMailDlg, CDialog)
    //{{AFX_MSG_MAP(CEMailDlg)
    ON_BN_CLICKED(IDC_TEST, OnTest)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CEMailDlg message handlers

void CEMailDlg::OnTest()
{
    CSimpleMAPI t_SimpleMAPI;
    CString t_strVideoFile;

    UpdateData(TRUE);

    if ( m_AttachVideo )
        GetMostRecentVideoFile(t_strVideoFile);

    t_SimpleMAPI.QuickSendMail(m_To, m_Cc, m_Subject, m_Message,
t_strVideoFile);

    return ;
}

void CEMailDlg::OnCancel()
{
    if ( AfxMessageBox(IDS_CANCELAREYOUSURE, MB_YESNO |
MB_ICONQUESTION) == IDNO )
        return;

    CDialog::OnCancel();
}

```

```

BOOL CEMailDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    CString t_strDateTimeFormat("HH':'mm");
    m_DurationCtrl.SetFormat(t_strDateTimeFormat);

    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CEMailDlg::OnOK()
{
    UpdateData(TRUE);

    CDialog::OnOK();
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
// Error Management.cpp
//
//
// Copyright (C) BKLK 2002.
// All rights reserved.
//
//
// DESCRIPTION
//      General routines for managing errors
//
//
// Change Log:
// 11-Mar-02 (lck)      - Created
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

#include "stdafx.h"

bool NProcessMessage(long lLineNumber, LPSTR szFilename, LPSTR
szMessage, int iMessageType)
{
    bool t_bReturn = false;

    SendMessage(HWND_BROADCAST, NMESSAGE, 0, 0);

    return t_bReturn;
}

// EventDialog.cpp : implementation file
//

#include "stdafx.h"

```

```

#include "Project Nalay.h"

#include "SimpleVid o.h"
#include "DirectPlay.h"

#include "EventDialog.h"
#include "resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CEventDialog dialog

CEventDialog::CEventDialog(CWnd* pParent /*=NULL*/)
: CDialog(CEventDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CEventDialog)
    m_Label = _T("");
    m_Friday = FALSE;
    m_LabelTitle = _T("");
    m_Monday = FALSE;
    m_Saturday = FALSE;
    m_StartTime = 0;
    m_StartTimeTitle = _T("");
    m_Sunday = FALSE;
    m_Thursday = FALSE;
    m_Tuesday = FALSE;
    m_Wednesday = FALSE;
    m_EndTime = 0;
    //}}AFX_DATA_INIT

    m_bNewEvent = true;
}

void CEventDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CEventDialog)
    DDX_Control(pDX, IDC_LABEL, m_LabelCtrl);
    DDX_Control(pDX, IDC_ENDTIME, m_EndTimeCtrl);
    DDX_Control(pDX, IDC_STARTTIME, m_StartTimeCtrl);
    DDX_Control(pDX, IDC_LISTVIEWSTYLE, m_ListViewStyle);
    DDX_Control(pDX, IDC_TYPESOFACTIONS, m_TypesOfActions);
    DDX_Control(pDX, IDC_ACTIONLIST, m_ActionList);
    DDX_Text(pDX, IDC_LABEL, m_Label);
    DDX_Check(pDX, IDC_FRIDAY, m_Friday);
    DDX_Text(pDX, IDC_LABELTITLE, m_LabelTitle);
    DDX_Check(pDX, IDC_MONDAY, m_Monday);

```

```

DDX_Check(pDX, IDC_SATURDAY, m_Saturday);
DDX_DateTimeCtrl(pDX, IDC_STARTTIME, m_StartTime);
DDX_Text(pDX, IDC_STARTTIMETITLE, m_StartTimeTitle);
DDX_Check(pDX, IDC_SUNDAY, m_Sunday);
DDX_Check(pDX, IDC_THURSDAY, m_Thursday);
DDX_Check(pDX, IDC_TUESDAY, m_Tuesday);
DDX_Check(pDX, IDC_WEDNESDAY, m_Wednesday);
DDX_DateTimeCtrl(pDX, IDC_ENDTIME, m_EndTime);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CEventDialog, CDialog)
//{{AFX_MSG_MAP(CEventDialog)
ON_BN_CLICKED(IDC_ADD, OnAdd)
ON_CBN_SELCHANGE(IDC_LISTVIEWSTYLE, OnSelchangeListviewstyle)
ON_BN_CLICKED(IDC_EDIT, OnEdit)
ON_BN_CLICKED(IDC_DELETE, OnDelete)
ON_NOTIFY(NM_DBLCLK, IDC_ACTIONLIST, OnDbldclkActionlist)
ON_BN_CLICKED(IDC_SELECT_CLEAR_DAYS, OnSelectClearDays)
ON_BN_CLICKED(IDC_SELECT_FULLWEEK, OnSelectFullweek)
ON_BN_CLICKED(IDC_SELECT_WEEKDAYS, OnSelectWeekdays)
ON_BN_CLICKED(IDC_SELECT_WEEKEND, OnSelectWeekend)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CEventDialog message handlers

BOOL CEventDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    CString t_str;
    CWnd* t_wnd;

    m_LargeImageList.Create(IDB_ACTIONS_LARGE_ICONS, 32, 1, RGB(255,
255, 255));
    m_SmallImageList.Create(IDB_ACTIONS_SMALL_ICONS, 16, 1, RGB(255,
255, 255));

    m_ActionList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    m_ActionList.SetImageList(&m_SmallImageList, LVSIL_SMALL);

    // Seed predetermined values into actions listbox
    m_TypesOfActions.Clear();
    t_str.LoadString(IDS_ACTIONDESC_EMAIL);
    m_TypesOfActions.AddString(t_str);
    t_str.LoadString(IDS_ACTIONDESC_PHONE);
    m_TypesOfActions.AddString(t_str);
    t_str.LoadString(IDS_ACTIONDESC_RECORDVIDEO);
    m_TypesOfActions.AddString(t_str);
    t_str.LoadString(IDS_ACTIONDESC_X10);
    m_TypesOfActions.AddString(t_str);
    t_str.LoadString(IDS_ACTIONDESC_AUDIO);
    m_TypesOfActions.AddString(t_str);

```

```

// Select default action type selection
m_TypesOfActions.SetCurSel(ACTIONTYPE_EMAIL);

// Set default list view and associated combo box value
SetViewType(LVS_REPORT);
m_ActionList.ModifyStyle(NULL, LVS_NOSORTHEADER | LVS_SINGLESEL);
m_ListViewStyle.SetCurSel(3);

// Set up columns
t_str.LoadString(IDS_ACTIONTYPE);
m_ActionList.InsertColumn(0, t_str);
m_ActionList.SetColumnWidth(0, 140);

t_str.LoadString(IDS_DETAILS);
m_ActionList.InsertColumn(1, t_str);
m_ActionList.SetColumnWidth(1, 350);

// Set up labels and controls depending on whether this is a
// scheduled event or an alarm
switch ( m_EventInfo.m_EventType )
{
case EVENTTYPE_SCHEDULEDEVENT:
    t_str.LoadString(IDS_SCHEDULEDEVENT);
    SetWindowText(t_str);
    m_LabelTitle.LoadString(IDS_SCHEDULEDEVENTLABEL);
    m_StartTimeTitle.LoadString(IDS_SCHEDULEDETIME);
    t_wnd = GetDlgItem(IDC_ENDTIMETITLE);
    t_wnd->ShowWindow(SW_HIDE);
    t_wnd = GetDlgItem(IDC_ENDTIME);
    t_wnd->ShowWindow(SW_HIDE);
    break;
case EVENTTYPE_ALARM:
    t_str.LoadString(IDS_MOTIONDETECTIONALARM);
    SetWindowText(t_str);
    m_LabelTitle.LoadString(IDS_ALARMMLABEL);
    m_StartTimeTitle.LoadString(IDS_BEGINCHECKING);
    break;
}

// Initailize dialog variables
m_Label = m_EventInfo.m_Label ;
m_StartTime = m_EventInfo.m_StartTime ;
m_EndTime = m_EventInfo.m_EndTime ;
m_Sunday = m_EventInfo.m_Sunday ;
m_Monday = m_EventInfo.m_Monday ;
m_Tuesday = m_EventInfo.m_Tuesday ;
m_Wednesday = m_EventInfo.m_Wednesday ;
m_Thursday = m_EventInfo.m_Thursday ;
m_Friday = m_EventInfo.m_Friday ;
m_Saturday = m_EventInfo.m_Saturday ;

// Format time control
CString t_strDateTimeFormat("hh':'mm' 'tt");
m_StartTimeCtrl.SetFormat(t_strDateTimeFormat);
m_EndTimeCtrl.SetFormat(t_strDateTimeFormat);

```

```

    if ( m_bNewEvent )
    {
        // If it is a new alarm...
        if ( m_EventInfo.m_EventType == EVENTTYPE_ALARM )
        {
            // ... insert a record action
            CVideoRecordAction * t_VideoRecordAction = new
CVideoRecordAction ;
            m_EventInfo.m_Actions.Add(t_VideoRecordAction);
        }
    }
    else
    {
        // If this is an existing event, disable name
        m_LabelCtrl.EnableWindow(FALSE);
    }

    // Need to update the dialog to see new values for controls
    UpdateData(FALSE);
    UpdateListCtrl();

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CEventDialog::OnAdd()
{
    int t_nTypeOfAction;

    t_nTypeOfAction = m_TypesOfActions.GetCurSel();

    // Should be impossible
    if ( t_nTypeOfAction == -1 ) return;

    if ( m_EventInfo.m_Actions.AddAction(t_nTypeOfAction) )
        UpdateListCtrl();
}

void CEventDialog::OnOK()
{
    CString strMessage;

    UpdateData(TRUE);

    // If this is a new connection make sure that the label is unique
and valid
    if ( m_bNewEvent )
    {
        if ( !gm_Connections.IsEventLabelValid(m_ConnectionLabel,
m_Label) )
        {
            strMessage.LoadString(IDS_ERR_INVALIDLABEL);
            m_LabelCtrl.SetFocus();
            MessageBox(strMessage);
            return;
        }
    }
}

```

```

    }
}

m_EventInfo.m_Label = m_Label;
m_EventInfo.m_StartTime = m_StartTime ;
m_EventInfo.m_EndTime = m_EndTime ;
m_EventInfo.m_Sunday = m_Sunday ;
m_EventInfo.m_Monday = m_Monday;
m_EventInfo.m_Tuesday = m_Tuesday;
m_EventInfo.m_Wednesday = m_Wednesday;
m_EventInfo.m_Thursday = m_Thursday;
m_EventInfo.m_Friday = m_Friday;
m_EventInfo.m_Saturday = m_Saturday;

CDialog::OnOK();

}

void CEventDialog::OnSelchangeListViewstyle()
{
    int t_nCurSel = m_ListViewStyle.GetCurSel();

    switch ( t_nCurSel )
    {
    case 0:
        if (GetViewType() != LVS_ICON)
            SetViewType(LVS_ICON);
        break;
    case 1:
        if (GetViewType() != LVS_SMALLICON)
            SetViewType(LVS_SMALLICON);
        break;
    case 2:
        if (GetViewType() != LVS_LIST)
            SetViewType(LVS_LIST);
        break;
    case 3:
        if (GetViewType() != LVS_REPORT)
            SetViewType(LVS_REPORT);
        break;
    }
}

}

BOOL CEventDialog::SetViewType(DWORD dwViewType)
{
    return(m_ActionList.ModifyStyle(LVS_TYPEMASK,dwViewType &
LVS_TYPEMASK));
}

DWORD CEventDialog::GetViewType()
{
    return(m_ActionList.GetStyle() & LVS_TYPEMASK);
}

bool CEventDialog::UpdateListCtrl()
{

```

```

bool t_bReturn = false;
int i;
CString t_str;

// Clear contents of control
m_ActionList.DeleteAllItems();

// Get array of connections
for ( i = 0; i < m_EventInfo.m_Actions.GetSize(); i++ )
{
    CActionInfo* t_ActionInfo = (CActionInfo* )
m_EventInfo.m_Actions[i];

    // Get description
    t_str.LoadString(IDS_ACTIONDESC_EMAIL + t_ActionInfo-
>m_ActionType );

    // Insert item
    m_ActionList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
                            i, t_str,
                            NULL, NULL, t_ActionInfo-
>m_ActionType,
                            NULL);

    // Set the remaining fields
    t_ActionInfo->GetActionSummary(t_str);
    m_ActionList.SetItemText(i, 1, t_str);
}

t_bReturn = true;
//Exit1:
return t_bReturn;
}

int CEventDialog::GetSelectedItem()
{
    int t_nItem = -1;

    // Get selected item - this is a single-selection list control
    POSITION t_pos = m_ActionList.GetFirstSelectedItemPosition();

    // Validate that an item was selected
    if (t_pos == NULL)
        goto Exit1;

    // Get the item number selected
    t_nItem = m_ActionList.GetNextSelectedItem(t_pos);

Exit1:
    return t_nItem;
}

void CEventDialog::OnEdit()
{
    int t_nActionIndex;

    // Get index of Action to edit

```

```

        t_nActionInd x = GetSelectedItem();

        // // Ensure that it is valid
        if ( t_nActionIndex < 0 )
            return;

        m_EventInfo.m_Actions.EditAction( t_nActionIndex );

        UpdateListCtrl();
    }

void CEventDialog::OnDelete()
{
    int t_nActionIndex;

    // Get index of Action to edit
    t_nActionIndex = GetSelectedItem();

    // // Ensure that it is valid
    if ( t_nActionIndex < 0 )
        return;

    m_EventInfo.m_Actions.DeleteAction( t_nActionIndex );

    UpdateListCtrl();
}

void CEventDialog::OnDblclkActionlist(NMHDR* pNMHDR, LRESULT* pResult)
{
    OnEdit();

    *pResult = 0;
}

void CEventDialog::OnCancel()
{
    if ( AfxMessageBox(IDS_CANCELAREYOUSURE, MB_YESNO |
MB_ICONQUESTION) == IDNO )
        return;

    CDialog::OnCancel();
}

void CEventDialog::OnSelectCleardays()
{
    UpdateData(TRUE);
    m_Sunday = FALSE;
    m_Monday = FALSE;
    m_Tuesday = FALSE;
    m_Wednesday = FALSE;
    m_Thursday = FALSE;
    m_Friday = FALSE;
    m_Saturday = FALSE;
    UpdateData(FALSE);
}

```

{

}

1

}

{

1

//

```
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
// CFeedbackErrorsView
```

```

IMPLEMENT_DYNCREATE(CFeedbackErrorsView, CEditView)

CFeedbackErrorsView::CFeedbackErrorsView()
{
}

CFeedbackErrorsView::~CFeedbackErrorsView()
{
}

BEGIN_MESSAGE_MAP(CFeedbackErrorsView, CEditView)
    //{AFX_MSG_MAP(CFeedbackErrorsView)
    // NOTE - the ClassWizard will add and remove mapping
    macros here.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CFeedbackErrorsView drawing

void CFeedbackErrorsView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CFeedbackErrorsView diagnostics

#ifdef _DEBUG
void CFeedbackErrorsView::AssertValid() const
{
    CEditView::AssertValid();
}

void CFeedbackErrorsView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CFeedbackErrorsView message handlers

void CFeedbackErrorsView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
        case NUPDATE_ERRORMESSAGE:
        case NUPDATE_DEBUGMESSAGE:
            CEdit& t_ctrlEdit = GetEditCtrl();

```

```

        LPSTR t_szMessage;

        t_szMessage = (LPSTR) pHint;
        m_strText += t_szMessage ;
        m_strText += "\r\n";

        t_ctrlEdit.SetWindowText(m_strText);
        break;
    };
}
// FeedbackFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "FeedbackFrame.h"
#include "FeedbackErrorsView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;

////////////////////////////////////
/////
// CFeedbackFrame

IMPLEMENT_DYNCREATE(CFeedbackFrame, CMDIChildWnd)

CFeedbackFrame::CFeedbackFrame()
{
}

CFeedbackFrame::~CFeedbackFrame()
{
}

BEGIN_MESSAGE_MAP(CFeedbackFrame, CMDIChildWnd)
    //{AFX_MSG_MAP(CFeedbackFrame)
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_CLOSE()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CFeedbackFrame message handlers
/*
BOOL CFeedbackFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // create a splitter with 2 rows, 1 column

```

```

    if (!m_wndSplitter.CreateStatic(this, 2, 1))
    {
        TRACE0("Failed to CreateStaticSplitter\n");
        return FALSE;
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
        pContext->m_pNewViewClass, CSize(130, 50), pContext))
    {
        TRACE0("Failed to create first pane\n");
        return FALSE;
    }

    // add the second splitter pane - an input view in row 1
    if (!m_wndSplitter.CreateView(1, 0,
        RUNTIME_CLASS(CFeedbackErrorsView), CSize(0, 0), pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(1,0));

    return TRUE;
}
*/

int CFeedbackFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_FEEDBACK))
    {
        return FALSE;        // fail to create
    }

    return 0;
}

BOOL CFeedbackFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.x = gm_Layout.m_rectFeedbackWindow.left;
    cs.y = gm_Layout.m_rectFeedbackWindow.top;
    cs.cx = gm_Layout.m_rectFeedbackWindow.Width();
    cs.cy = gm_Layout.m_rectFeedbackWindow.Height();

    return CMDIChildWnd::PreCreateWindow(cs);
}

void CFeedbackFrame::OnDestroy()
{
    CMDIChildWnd::OnDestroy();
}

```

```

        GetWindowRect(&gm_Layout.m_rectFeedbackWindow);
        GetParent()->ScreenToClient(&gm_Layout.m_rectFeedbackWindow);
    }

void CFeedbackFrame::OnClose()
{
    CProjectNalayApp * t_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    ASSERT ( t_pProjectNalayApp != NULL );

    if ( t_pProjectNalayApp != NULL )
        t_pProjectNalayApp->CloseFeedbackListView();

    gm_Layout.m_bFeedbackWindowOpen = FALSE;

    CMDIChildWnd::OnClose();
}
// FeedbackInfoDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "FeedbackInfoDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CFeedbackInfoDlg dialog

CFeedbackInfoDlg::CFeedbackInfoDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFeedbackInfoDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFeedbackInfoDlg)
    m_Description = _T("");
    m_Location = _T("");
    m_Message = _T("");
    m_Time = _T("");
    //}}AFX_DATA_INIT
}

void CFeedbackInfoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFeedbackInfoDlg)
    DDX_Text(pDX, IDC_DETAILS, m_Description);
    DDX_Text(pDX, IDC_LOCATION, m_Location);
    DDX_Text(pDX, IDC_MESSAGE, m_Message);
    DDX_Text(pDX, IDC_TIME, m_Time);
    //}}AFX_DATA_MAP

```

```

}

BEGIN_MESSAGE_MAP(CFeedbackInfoDlg, CDialog)
    //{AFX_MSG_MAP(CFeedbackInfoDlg)
        // NOTE: the ClassWizard will add message map macros here
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CFeedbackInfoDlg message handlers
// FeedbackListView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "FeedbackListView.h"
#include "FeedbackInfoDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CFeedbackListView

IMPLEMENT_DYNCREATE(CFeedbackListView, CListView)

CFeedbackListView::CFeedbackListView()
{
}

CFeedbackListView::~CFeedbackListView()
{
}

BEGIN_MESSAGE_MAP(CFeedbackListView, CListView)
    //{AFX_MSG_MAP(CFeedbackListView)
        ON_NOTIFY_REFLECT(NM_DBLCLK, OnDbclk)
        ON_COMMAND(ID_FEEDBACK_DEBUG, OnFeedbackDebug)
        ON_UPDATE_COMMAND_UI(ID_FEEDBACK_DEBUG, OnUpdateFeedbackDebug)
        ON_COMMAND(ID_FEEDBACK_ERROR, OnFeedbackError)
        ON_UPDATE_COMMAND_UI(ID_FEEDBACK_ERROR, OnUpdateFeedbackError)
        ON_COMMAND(ID_FEEDBACK_STATUS, OnFeedbackStatus)
        ON_UPDATE_COMMAND_UI(ID_FEEDBACK_STATUS, OnUpdateFeedbackStatus)
        ON_COMMAND(ID_FEEDBACK_WARNING, OnFeedbackWarning)
        ON_UPDATE_COMMAND_UI(ID_FEEDBACK_WARNING,
OnUpdateFeedbackWarning)
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

/////////////////////////////////////////////////////////////////
/////
// CFeedbackListView drawing

void CFeedbackListView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////
/////
// CFeedbackListView diagnostics

#ifdef _DEBUG
void CFeedbackListView::AssertValid() const
{
    CListView::AssertValid();
}

void CFeedbackListView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////
/////
// CFeedbackListView message handlers

void CFeedbackListView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    // Set up icons
    m_LargeImageList.Create(IDB_FEEDBACKLARGEICONS, 32, 1, RGB(255,
255, 255));
    m_SmallImageList.Create(IDB_FEEDBACKSMALLICONS, 16, 1, RGB(255,
255, 255));
    // m_StateImageList.Create(IDB_CONNECTIONSSTATEICONS, 8, 1, RGB(255,
0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    // t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

    // Set up columns
    t_strItem.LoadString(IDS_MESSAGE);
    t_ctlList.InsertColumn(0, t_strItem);
    SetColumnWidth(0, 110);
    t_strItem.LoadString(IDS_DESCRIPTION);
    t_ctlList.InsertColumn(1, t_strItem);
    SetColumnWidth(1, 260);

```

```

        t_strItem.LoadString(IDS_TIME);
        t_ctlList.InsertColumn(2, t_strItem);
        SetColumnWidth(2, 120);
        t_strItem.LoadString(IDS_LOCATION);
        t_ctlList.InsertColumn(3, t_strItem);
        SetColumnWidth(3, 80);
    }

void CFeedbackListView::AddMessage(long lLineNumber, LPSTR szFilename,
int nType, CString strMessage, CString strDescription)
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem;
    CString t_strLocation;
    TCHAR t_szJustFileName[_MAX_PATH];
    TCHAR t_szFileName[_MAX_PATH];
    CTime t_curTime;
    CString t_strTime;

    t_curTime = CTime::GetCurrentTime();
    t_strTime = t_curTime.Format("%c");

    // Filter messages
    if ( nType == FEEDBACKMESSAGETYPE_STATUS &&
!gm_AppConfig.m_ShowStatusFeedback )
        return;
    if ( nType == FEEDBACKMESSAGETYPE_WARNING &&
!gm_AppConfig.m_ShowWarningFeedback )
        return;
    if ( nType == FEEDBACKMESSAGETYPE_ERROR &&
!gm_AppConfig.m_ShowErrorFeedback )
        return;
    if ( nType == FEEDBACKMESSAGETYPE_DEBUG &&
!gm_AppConfig.m_ShowDebugFeedback )
        return;

    // Get just the filename
    lstrcpy(t_szFileName, szFilename);
    lstrcpy(t_szJustFileName, PathFindFileName(t_szFileName));
    PathRemoveExtension(t_szJustFileName);

    t_nItem = t_ctlList.GetItemCount();

    // Insert item
    t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
                        t_nItem , strMessage,
                        NULL, NULL, nType,
                        NULL);

    // Set the remaining fields
    t_ctlList.SetItemText(t_nItem, 1, strDescription);
    t_strLocation.Format("%s, %ld", t_szJustFileName, lLineNumber);
    t_ctlList.SetItemText(t_nItem, 2, t_strTime);
    t_ctlList.SetItemText(t_nItem, 3, t_strLocation);

    t_ctlList.EnsureVisible(t_nItem, FALSE);
}

```

```

}

BOOL CFeedbackListView::SetViewType(DWORD dwViewType)
{
    return(ModifyStyle(LVS_TYPEMASK, dwViewType & LVS_TYPEMASK));
}

DWORD CFeedbackListView::GetViewType()
{
    return(GetStyle() & LVS_TYPEMASK);
}

BOOL CFeedbackListView::SetColumnWidth(int Column, int Width)
{
    CListCtrl& t_ctlList = GetListCtrl();
    LVCOLUMN t_lvColumn;

    t_lvColumn.mask = LVCF_WIDTH;
    t_lvColumn.cx = Width;
    return t_ctlList.SetColumn(Column, &t_lvColumn);
}

void CFeedbackListView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem ;
    CString t_strMessage, t_strDescription, t_strLocation;
    CFeedbackInfoDlg t_FeedbackInfoDlg;

    // Get item selected
    t_nItem = GetSelectedItem();
    if ( t_nItem < 0 )
        goto Exit1;

    // Get info related to selected feedback item
    t_FeedbackInfoDlg.m_Message = t_ctlList.GetItemText(t_nItem, 0);
    t_FeedbackInfoDlg.m_Description = t_ctlList.GetItemText(t_nItem,
1);
    t_FeedbackInfoDlg.m_Time = t_ctlList.GetItemText(t_nItem, 2);
    t_FeedbackInfoDlg.m_Location = t_ctlList.GetItemText(t_nItem, 3);

    t_FeedbackInfoDlg.DoModal();

Exit1:
    *pResult = 0;
}

int CFeedbackListView::GetSelectedItem()
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem = -1;

    // Get selected item - this is a single-selection list control
    POSITION t_pos = t_ctlList.GetFirstSelectedItemPosition();

    // Validate that an item was selected

```

```

        if (t_pos == NULL)
        {
            NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_WARNING, "",
IDS_ERR_NOITEMSELECTED);
            goto Exit1;
        }

        // Get the item number selected
        t_nItem = t_ctlList.GetNextSelectedItem(t_pos);

Exit1:
    return t_nItem;
}

BOOL CFeedbackListView::PreCreateWindow(CREATESTRUCT& cs)
{
    // Default to report view
    cs.style |= LVS_REPORT | LVS_NOSORTHEADER | LVS_SINGLESEL;

    return CListView::PreCreateWindow(cs);
}

void CFeedbackListView::OnFeedbackDebug()
{
    gm_AppConfig.m_ShowDebugFeedback =
!gm_AppConfig.m_ShowDebugFeedback ;
}

void CFeedbackListView::OnUpdateFeedbackDebug(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_AppConfig.m_ShowDebugFeedback );
}

void CFeedbackListView::OnFeedbackError()
{
    gm_AppConfig.m_ShowErrorFeedback =
!gm_AppConfig.m_ShowErrorFeedback ;
}

void CFeedbackListView::OnUpdateFeedbackError(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_AppConfig.m_ShowErrorFeedback );
}

void CFeedbackListView::OnFeedbackStatus()
{
    gm_AppConfig.m_ShowStatusFeedback =
!gm_AppConfig.m_ShowStatusFeedback ;
}

void CFeedbackListView::OnUpdateFeedbackStatus(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_AppConfig.m_ShowStatusFeedback );
}

void CFeedbackListView::OnFeedbackWarning()

```

```

{
    gm_AppConfig.m_ShowWarningFeedback =
!gm_AppConfig.m_ShowWarningFeedback ;
}

void CFeedbackListView::OnUpdateFeedbackWarning(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_AppConfig.m_ShowWarningFeedback );
}
// FeedbackStatusView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "FeedbackStatusView.h"
#include "Main Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CFeedbackStatusView

IMPLEMENT_DYNCREATE(CFeedbackStatusView, CEditView)

CFeedbackStatusView::CFeedbackStatusView()
{
}

CFeedbackStatusView::~CFeedbackStatusView()
{
}

BEGIN_MESSAGE_MAP(CFeedbackStatusView, CEditView)
    //{AFX_MSG_MAP(CFeedbackStatusView)
    // NOTE - the ClassWizard will add and remove mapping
    macros here.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CFeedbackStatusView drawing

void CFeedbackStatusView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////

```

```

// CFeedbackStatusView diagnostics

#ifdef _DEBUG
void CFeedbackStatusView::AssertValid() const
{
    CEditView::AssertValid();
}

void CFeedbackStatusView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CFeedbackStatusView message handlers

void CFeedbackStatusView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
        case NUPDATE_STATUSMESSAGE:
            CEdit& t_ctrlEdit = GetEditCtrl();
            LPSTR t_szMessage;

            t_szMessage = (LPSTR) pHint;
            m_strText += t_szMessage ;
            m_strText += "\r\n";

            t_ctrlEdit.SetWindowText(m_strText);
            break;
    };
}
// InstantMessengerConversationView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "InstantMessengerConversationView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CInstantMessengerConversationView

```

```

IMPLEMENT_DYNCREATE(CInstantMessengerConversationView, CListView)

CInstantMessengerConversationView::CInstantMessengerConversationView()
{
}

CInstantMessengerConversationView::~CInstantMessengerConversationView()
{
}

BEGIN_MESSAGE_MAP(CInstantMessengerConversationView, CListView)
    //{AFX_MSG_MAP(CInstantMessengerConversationView)
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CInstantMessengerConversationView drawing

void CInstantMessengerConversationView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CInstantMessengerConversationView diagnostics

#ifdef _DEBUG
void CInstantMessengerConversationView::AssertValid() const
{
    CListView::AssertValid();
}

void CInstantMessengerConversationView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CInstantMessengerConversationView message handlers

void CInstantMessengerConversationView::OnUpdate(CView* pSender, LPARAM
lHint, CObject* pHint)
{
    CString t_str ;
    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strText;

    // Next item to add
    int t_nItem = t_ctlList.GetItemCount();

```

```

switch ( lHint )
{
case NUPDATE_IMMESSAGE RECEIVED:
{
CConnectionMsg* t_ConnectionMsg;

t_ConnectionMsg = (CConnectionMsg*) pHint;

// Update window
// Insert item
t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
t_nItem, t_ConnectionMsg-
>m_szLabel,

NULL, NULL, 1,
NULL);

// Set the remaining fields
t_ctlList.SetItemText(t_nItem, 1, t_ConnectionMsg-
>m_szMessage);
}
break;

case NUPDATE_IMMESSAGE SENT:
{
CConnectionMsg* t_ConnectionMsg;

t_ConnectionMsg = (CConnectionMsg*) pHint;

// Update window
// Insert item
t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
t_nItem, t_ConnectionMsg-
>m_szLabel,

NULL, NULL, 0,
NULL);

// Set the remaining fields
t_ctlList.SetItemText(t_nItem, 1, t_ConnectionMsg-
>m_szMessage);
}
break;
};

t_ctlList.EnsureVisible(t_nItem, FALSE);
}

BOOL CInstantMessengerConversationView::PreCreateWindow(CREATESTRUCT&
cs)
{
// Default to report view
cs.style |= LVS_REPORT | LVS_NOSORTHEADER | LVS_SINGLESEL;

return CListView::PreCreateWindow(cs);
}

void CInstantMessengerConversationView::OnInitialUpdate()

```

```

{
    CListView::OnInitialUpdate();
    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    // Set up icons
    m_LargeImageList.Create(IDB_INSTANTMESSENGERLARGEICONS, 32, 1,
    RGB(255, 255, 255));
    m_SmallImageList.Create(IDB_INSTANTMESSENGERSMALLICONS, 16, 1,
    RGB(255, 255, 255));
    // m_StateImageList.Create(IDB_CONNECTIONSSTATEICONS, 8, 1, RGB(255,
    0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    // t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

    // Set up columns
    t_strItem.LoadString(IDS_SOURCE);
    t_ctlList.InsertColumn(0, t_strItem);
    SetColumnWidth(0, 150);
    t_strItem.LoadString(IDS_MESSAGE);
    t_ctlList.InsertColumn(1, t_strItem);
    SetColumnWidth(1, 500);
}

BOOL CInstantMessengerConversationView::SetColumnWidth(int Column, int
Width)
{
    CListCtrl& t_ctlList = GetListCtrl();
    LV_COLUMN t_lvColumn;

    t_lvColumn.mask = LVCF_WIDTH;
    t_lvColumn.cx = Width;
    return t_ctlList.SetColumn(Column, &t_lvColumn);
}

// InstantMessengerDoc.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "InstantMessengerDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

```

```

extern CLayout gm_Layout;
extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CInstantMessengerDoc

IMPLEMENT_DYNCREATE(CInstantMessengerDoc, CDocument)

CInstantMessengerDoc::CInstantMessengerDoc()
{
}

BOOL CInstantMessengerDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    return TRUE;
}

CInstantMessengerDoc::~CInstantMessengerDoc()
{
    gm_Connections.Disconnect(m_ConnectionLabel);
}

BEGIN_MESSAGE_MAP(CInstantMessengerDoc, CDocument)
    ///{AFX_MSG_MAP(CInstantMessengerDoc)
    ///}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CInstantMessengerDoc diagnostics

#ifdef _DEBUG
void CInstantMessengerDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CInstantMessengerDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CInstantMessengerDoc serialization

void CInstantMessengerDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
}

```

```

        else
        {
            // TODO: add loading code here
        }
    }

////////////////////////////////////
////////
// CInstantMessengerDoc commands

BOOL CInstantMessengerDoc::CanCloseFrame(CFrameWnd* pFrame)
{
    gm_Layout.DeleteLayoutConnection(m_ConnectionLabel);

    return CDocument::CanCloseFrame(pFrame);
}

BOOL CInstantMessengerDoc::IsModified()
{
    return FALSE;
}

// InstantMessengerEntryView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "Main Doc.h"
#include "InstantMessengerEntryView.h"
#include "InstantMessengerDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CConnectionsArray gm_Connections;

////////////////////////////////////
////////
// CInstantMessengerEntryView

IMPLEMENT_DYNCREATE(CInstantMessengerEntryView, CEditView)

CInstantMessengerEntryView::CInstantMessengerEntryView()
{
}

CInstantMessengerEntryView::~CInstantMessengerEntryView()
{
}

```

```

BEGIN_MESSAGE_MAP(CInstantMessengerEntryView, CEditView)
    //{AFX_MSG_MAP(CInstantMessengerEntryView)
    ON_COMMAND(ID_CONNECTION_CONNECT, OnConnectionConnect)
    ON_COMMAND(ID_CONNECTION_DISCONNECT, OnConnectionDisconnect)
    ON_WM_KEYUP()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CInstantMessengerEntryView drawing

void CInstantMessengerEntryView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CInstantMessengerEntryView diagnostics

#ifdef _DEBUG
void CInstantMessengerEntryView::AssertValid() const
{
    CEditView::AssertValid();
}

void CInstantMessengerEntryView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CInstantMessengerEntryView message handlers

void CInstantMessengerEntryView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
        case NUPDATE_CLOSECONNECTIONWINDOW:
/*
            CString* t_str = (CString*) pHint;
            CString t_strLabel = t_str->GetBuffer(t_str-
>GetLength()+1);

            // See if connection that disconnected matches this one
            if ( t_strLabel == m_ConnectionLabel)
                GetParent()->CloseWindow();
*/
            break;
    };
}

```

```

}

bool CInstantMessengerEntryView::GetConnectionLabel(CString&
strConnectionLabel)
{
    CInstantMessengerDoc * t_InstantMessengerDoc =
(CInstantMessengerDoc * ) GetDocument();
    bool t_bReturn = false;

    if ( t_InstantMessengerDoc == NULL )
        goto Exit1;

    strConnectionLabel = t_InstantMessengerDoc->m_ConnectionLabel;

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

void CInstantMessengerEntryView::OnConnectionConnect()
{
    CString t_strConnectionLabel;

    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;

    gm_Connections.Connect(t_strConnectionLabel, m_hWnd);
}

void CInstantMessengerEntryView::OnConnectionDisconnect()
{
    CString t_strConnectionLabel;

    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;

    gm_Connections.Disconnect(t_strConnectionLabel);
}

void CInstantMessengerEntryView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT
nFlags)
{
    CString t_strConnectionLabel;
    CConnectionMsg t_ConnectionMsg;

    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;

    // Check if the Enter key was pressed
    if ( nChar == 13 )
    {
        CEdit& t_ctrlEdit = GetEditCtrl();
        CString t_strMessage;
        CDocument* t_pDoc = GetDocument();

        // Get message
        t_ctrlEdit.GetWindowText(t_strMessage);
    }
}

```

```

        t_strMessage.Remove(13);
        t_strMessage.Remove(10);

        // Setup message handler
        t_ConnectionMsg.m_nMessage Type =
CONNECTIONMESSAGE_TYPE_IMMEDIATE;
        t_ConnectionMsg.SetMessage(t_strMessage);
        t_ConnectionMsg.SetLabel(t_strConnectionLabel);
        t_ConnectionMsg.m_bLocal = true;

        // Update conversation window
        if ( t_pDoc )
            t_pDoc->UpdateAllViews(NULL, NUUPDATE_IMMEDIATESENT,
(CObject *) &t_ConnectionMsg);

        // Send message to other connections
        gm_Connections.SendMessage(t_strConnectionLabel,
t_strMessage);

        // Clear out window
        t_strMessage.Empty();
        t_ctrlEdit.SetWindowText(t_strMessage);
    }

    CEditView::OnKeyUp(nChar, nRepCnt, nFlags);
}

// InstantMessengerFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "InstantMessengerFrame.h"
#include "InstantMessengerEntryView.h"
#include "InstantMessengerDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW

#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;

////////////////////////////////////
/////
// CInstantMessengerFrame

IMPLEMENT_DYNCREATE(CInstantMessengerFrame, CMDIChildWnd)

CInstantMessengerFrame::CInstantMessengerFrame()
{
}

CInstantMessengerFrame::~CInstantMessengerFrame()
{
}

```

```

}

BEGIN_MESSAGE_MAP(CInstantMessengerFrame, CMDIChildWnd)
//{{AFX_MSG_MAP(CInstantMessengerFrame)
    ON_WM_CREATE()
    ON_WM_DESTROY()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CInstantMessengerFrame message handlers

BOOL CInstantMessengerFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // create a splitter with 2 rows, 1 column
    if (!m_wndSplitter.CreateStatic(this, 2, 1))
    {
        TRACE0("Failed to CreateStaticSplitter\n");
        return FALSE;
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
        pContext->m_pNewViewClass, CSize(130, 150), pContext))
    {
        TRACE0("Failed to create first pane\n");
        return FALSE;
    }

    // add the second splitter pane - an input view in row 1
    if (!m_wndSplitter.CreateView(1, 0,
        RUNTIME_CLASS(CInstantMessengerEntryView), CSize(0, 0),
pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(1,0));

    return TRUE;
}

int CInstantMessengerFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_INSTANTMESSENGER))
    {

```

```

        return FALSE;        // fail to create
    }

    return 0;
}

BOOL CInstantMessengerFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    CRect t_RectWindow(25, 25, 400, 400);
    CString t_strConnectionLabel;
    CInstantMessengerDoc * t_InstantMessengerDoc;

    t_InstantMessengerDoc = ( CInstantMessengerDoc * )
    GetActiveDocument();

    if ( t_InstantMessengerDoc != NULL )
    {
        t_strConnectionLabel = t_InstantMessengerDoc-
        >m_ConnectionLabel;

        if (
gm_LayOut.GetConnectionWindowRect(t_strConnectionLabel, t_RectWindow )
        )
        {
            cs.x = t_RectWindow.left;
            cs.y = t_RectWindow.top;
            cs.cx = t_RectWindow.Width();
            cs.cy = t_RectWindow.Height();
        }
    }

    return CMDIChildWnd::PreCreateWindow(cs);
}

void CInstantMessengerFrame::OnDestroy()
{
    CMDIChildWnd::OnDestroy();

    CRect t_RectWindow;
    CString t_strConnectionLabel;
    CInstantMessengerDoc * t_InstantMessengerDoc;

    t_InstantMessengerDoc = ( CInstantMessengerDoc * )
    GetActiveDocument();

    if ( t_InstantMessengerDoc != NULL )
    {
        t_strConnectionLabel = t_InstantMessengerDoc-
        >m_ConnectionLabel;

        GetWindowRect(&t_RectWindow);
        GetParent()->ScreenToClient(&t_RectWindow);

        gm_LayOut.SetConnectionWindowRect(t_strConnectionLabel,
t_RectWindow );
    }
}

```

```

// It mListDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "ItemListDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CItemListDlg dialog

CItemListDlg::CItemListDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CItemListDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CItemListDlg)
    //}}AFX_DATA_INIT
}

void CItemListDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CItemListDlg)
    DDX_Control(pDX, IDC_ITEMLIST, m_ItemListCtrl);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CItemListDlg, CDialog)
    //{{AFX_MSG_MAP(CItemListDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CItemListDlg message handlers

BOOL CItemListDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    for (int i = 0; i < m_parrItems->GetSize(); i++ )
        m_ItemListCtrl.AddString(m_parrItems->ElementAt(i));

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}
// LocalVideoDoc.cpp : implementation file

```

```

//

#include "stdafx.h"
#include "Project Nalay.h"
#include "LocalVideoDoc.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;
extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CLocalVideoDoc

IMPLEMENT_DYNCREATE(CLocalVideoDoc, CDocument)

CLocalVideoDoc::CLocalVideoDoc()
{
    m_hWndVideoView = NULL;
}

BOOL CLocalVideoDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    return TRUE;
}

CLocalVideoDoc::~CLocalVideoDoc()
{
    gm_Connections.Disconnect(m_ConnectionLabel);
}

BEGIN_MESSAGE_MAP(CLocalVideoDoc, CDocument)
    //{AFX_MSG_MAP(CLocalVideoDoc)
    // NOTE - the ClassWizard will add and remove mapping
    macros here.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CLocalVideoDoc diagnostics

#ifdef _DEBUG
void CLocalVideoDoc::AssertValid() const
{
    CDocument::AssertValid();
}

```

```

}

void CLocalVideoDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////
////////
// CLocalVideoDoc serialization

void CLocalVideoDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

////////////////////////////////////
////////
// CLocalVideoDoc commands

void CLocalVideoDoc::OnCloseDocument()
{
    CDocument::OnCloseDocument();
}

BOOL CLocalVideoDoc::CanCloseFrame(CFrameWnd* pFrame)
{
    gm_Layout.DeleteLayoutConnection(m_ConnectionLabel);

    return CDocument::CanCloseFrame(pFrame);
}
// LocalVideoEventsView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "LocalVideoEventsView.h"
#include "LocalVideoDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;

```

```

#endif

extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CLocalVideoEventsView

IMPLEMENT_DYNCREATE(CLocalVideoEventsView, CListView)

CLocalVideoEventsView::CLocalVideoEventsView()
{
}

CLocalVideoEventsView::~CLocalVideoEventsView()
{
}

BEGIN_MESSAGE_MAP(CLocalVideoEventsView, CListView)
    //{AFX_MSG_MAP(CLocalVideoEventsView)
    ON_COMMAND(ID_CONNECTION_NEW_ALARM, OnConnectionNewAlarm)
    ON_COMMAND(ID_CONNECTION_NEW_EVENT, OnConnectionNewEvent)
    ON_COMMAND(ID_CONNECTION_CONNECT, OnConnectionConnect)
    ON_COMMAND(ID_CONNECTION_DISCONNECT, OnConnectionDisconnect)
    ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLargeIcons)
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
    ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallIcons)
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
    ON_NOTIFY_REFLECT(NM_DBLCLK, OnDbclk)
    ON_COMMAND(ID_CONNECTION_DELETE_EVENT, OnConnectionDeleteEvent)
    ON_WM_CONTEXTMENU()
    ON_COMMAND(ID_LOCALVIDEO_PAUSE, OnLocalvideoPause)
    ON_COMMAND(ID_LOCALVIDEO_PLAY, OnLocalvideoPlay)
    ON_COMMAND(ID_LOCALVIDEO_STOP, OnLocalvideoStop)
    ON_COMMAND(ID_LOCALVIDEO_RECORD, OnLocalvideoRecord)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_PAUSE,
OnUpdateLocalvideoPause)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_RECORD,
OnUpdateLocalvideoRecord)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_PLAY, OnUpdateLocalvideoPlay)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_STOP, OnUpdateLocalvideoStop)
    ON_COMMAND(ID_LOCALVIDEO_MOTIONDETECTION,
OnLocalvideoMotiondetection)
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_CONNECT,
OnUpdateConnectionConnect)
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_DISCONNECT,
OnUpdateConnectionDisconnect)
    ON_WM_DESTROY()
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_DELETE_EVENT,
OnUpdateConnectionDeleteEvent)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CLocalVideoEventsView drawing

```

```

void CLocalVideoEventsView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CLocalVideoEventsView diagnostics

#ifdef _DEBUG
void CLocalVideoEventsView::AssertValid() const
{
    CListView::AssertValid();
}

void CLocalVideoEventsView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CLocalVideoEventsView message handlers

void CLocalVideoEventsView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    m_LargeImageList.Create(IDB_EVENTS_LARGE_ICONS, 32, 1, RGB(255,
255, 255));
    m_SmallImageList.Create(IDB_EVENTS_SMALL_ICONS, 16, 1, RGB(255,
255, 255));
    // m_StateImageList.Create(IDB_CONNECTIONS_STATE_ICONS, 8, 1, RGB(255,
0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    // t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

    // Setup columns
    t_strItem.LoadString(IDS_LABEL);
    t_ctlList.InsertColumn(0, t_strItem);
    SetColumnWidth(0, 110);
    t_strItem.LoadString(IDS_TYPE);
    t_ctlList.InsertColumn(1, t_strItem);
    SetColumnWidth(1, 110);
    t_strItem.LoadString(IDS_DETAILS);
    t_ctlList.InsertColumn(2, t_strItem);
    SetColumnWidth(2, 450);

    // Initialize data - don't need to here because main app sends

```

```

        // an NUPDATE_REFRESH after the window is created
        // Otherwise the window attempts to get a connectionLabel
        // that does not yet get assigned to the CDocument
// UpdateListCtrl();
}

BOOL CLocalVideoEventsView::PreCreateWindow(CREATESTRUCT& cs)
{
    // Default to report view
    cs.style |= LVS_ICON | LVS_NOSORTHEADER | LVS_SINGLESEL;

    return CListView::PreCreateWindow(cs);
}

void CLocalVideoEventsView::OnConnectionNewAlarm()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc *)
    GetDocument();

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONALARM) )
        return ;

    if ( t_LocalVideoDoc == NULL )
        return;

    if ( gm_Connections.AddEvent(t_LocalVideoDoc->m_ConnectionLabel,
EVENTTYPE_ALARM) )
    {
        UpdateListCtrl();
        SendEventsArray();
    }
}

void CLocalVideoEventsView::OnConnectionNewEvent()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc *)
    GetDocument();

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONALARM) )
        return ;

    if ( t_LocalVideoDoc == NULL )
        return;

    if ( gm_Connections.AddEvent(t_LocalVideoDoc->m_ConnectionLabel,
EVENTTYPE_SCHEDULEDEVENT) )
    {
        UpdateListCtrl();
        SendEventsArray();
    }
}

bool CLocalVideoEventsView::GetConnectionLabel(CString&
strConnectionLabel)
{

```

```

        CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
        bool t_bReturn = false;

        if ( t_LocalVideoDoc == NULL )
            goto Exit1;

        strConnectionLabel = t_LocalVideoDoc->m_ConnectionLabel;

        t_bReturn = true;
Exit1:
        return t_bReturn;
    }

void CLocalVideoEventsView::OnConnectionConnect()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;

    gm_Connections.Connect(t_LocalVideoDoc->m_ConnectionLabel,
t_LocalVideoDoc->m_hWndVideoView);
}

void CLocalVideoEventsView::OnConnectionDisconnect()
{
    CString t_strConnectionLabel;
    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;
    gm_Connections.Disconnect(t_strConnectionLabel);
}

void CLocalVideoEventsView::OnViewLargeicons()
{
    if (GetViewType() != LVS_ICON)
        SetViewType(LVS_ICON);
}

void CLocalVideoEventsView::OnViewList()
{
    if (GetViewType() != LVS_LIST)
        SetViewType(LVS_LIST);
}

void CLocalVideoEventsView::OnViewSmallicons()
{
    if (GetViewType() != LVS_SMALLICON)
        SetViewType(LVS_SMALLICON);
}

void CLocalVideoEventsView::OnViewDetails()
{
    if (GetViewType() != LVS_REPORT)
        SetViewType(LVS_REPORT);
}

BOOL CLocalVideoEventsView::SetViewType(DWORD dwViewType)

```

[illegible]

```

        NULL);

        // Set the remaining fields
        t_EventInfo.GetEventTypeString(t_str);
        t_ctlList.SetItemText(i, 1, t_str);
        t_EventInfo.GetEventDetailString(t_str);
        t_ctlList.SetItemText(i, 2, t_str);
    }

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

void CLocalVideoEventsView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
    case NUPDATE_CONNECT:
        OnConnectionConnect();
        break;
    case NUPDATE_DISCONNECT:
        OnConnectionDisconnect() ;
        break;
    case NUPDATE_REFRESH:
        UpdateListCtrl();
        break;
    case NUPDATE_VIEWDETAILS:
        OnViewDetails();
        break;
    case NUPDATE_VIEWLARGEICONS:
        OnViewLargeicons() ;
        break;
    case NUPDATE_VIEWLIST:
        OnViewList() ;
        break;
    case NUPDATE_VIEWSMALLICONS:
        OnViewSmallicons();
        break;
    case NUPDATE_NEWALARM:
        OnConnectionNewAlarm();
        break;
    case NUPDATE_NEWSCHEDULEDEVENT:
        OnConnectionNewEvent();
        break;
    case NUPDATE_DELETEEVENT:
        OnConnectionDeleteEvent();
        break;
    // case NUPDATE_IMMESSAGERECEIVED:
    case DPN_MSGID_RECEIVE:
        {
            CConnectionMsg* t_pConnectionMsg;
            t_pConnectionMsg = (CConnectionMsg*) pHint;
            ReceiveEventsArray(t_pConnectionMsg);
        }
    }
}

```

```

        break;
    case DPN_MSGID_CREATE_PLAYER:
        SendEventsArray();
        break;
    };
}

void CLocalVideoEventsView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    OnEditEvent();

    *pResult = 0;
}

void CLocalVideoEventsView::OnEditEvent()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    CString t_strEventLabel;

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONALARM) )
        return ;

    if ( t_LocalVideoDoc == NULL )
        return;

    // Get label of selected item and delete it
    if ( GetSelectedEventLabel(t_strEventLabel) )
        if ( gm_Connections.EditEvent(t_LocalVideoDoc->
m_ConnectionLabel, t_strEventLabel) )
        {
            UpdateListCtrl();
            SendEventsArray();
        }
}

bool CLocalVideoEventsView::GetSelectedEventLabel(CString& strLabel)
{
    bool t_bReturn = false;

    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem ;

    // Get selected item - this is a single-selection list control
    t_nItem = GetSelectedEventItem();
    if ( t_nItem == -1 )
        goto Exit1;

    // Get the m_Connections array index for that item
    strLabel = t_ctlList.GetItemText(t_nItem, 0);

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

```

```

int CLocalVideoEventsView::GetSelectedEventItem()
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem = -1;

    // Get selected item - this is a single-selection list control
    POSITION t_pos = t_ctlList.GetFirstSelectedItemPosition();

    // Validate that an item was selected
    if (t_pos == NULL)
        goto Exit1;

    // Get the item number selected
    t_nItem = t_ctlList.GetNextSelectedItem(t_pos);

Exit1:
    return t_nItem;
}

void CLocalVideoEventsView::OnConnectionDeleteEvent()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
    GetDocument();
    CString t_strEventLabel;

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONALARM) )
        return ;

    if ( t_LocalVideoDoc == NULL )
        return;

    // Get label of selected item and delete it
    if ( GetSelectedEventLabel(t_strEventLabel) )
        if ( gm_Connections.DeleteEvent(t_LocalVideoDoc->
m_ConnectionLabel, t_strEventLabel) )
        {
            UpdateListCtrl();
            SendEventsArray();
        }
}

void CLocalVideoEventsView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    DisplayContextMenu(this, point, IDR_POPUP_LOCALVIDEO);
}

void CLocalVideoEventsView::OnLocalvideoPause()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
    GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_LOCALVIDEOPAUSE,
    NULL);
}

void CLocalVideoEventsView::OnLocalvideoPlay()

```

```

{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_LOCALVIDEOPLAY,
NULL);
}

void CLocalVideoEventsView::OnLocalvideoStop()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_LOCALVIDEOSTOP,
NULL);
}

void CLocalVideoEventsView::OnLocalvideoRecord()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_LOCALVIDEORECORD,
NULL);
}

void CLocalVideoEventsView::OnUpdateLocalvideoPause(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_PLAYING );
}

void CLocalVideoEventsView::OnUpdateLocalvideoRecord(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->SetCheck(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_RECORDING ||

gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_STREAMINGANDRECORDING );
}

void CLocalVideoEventsView::OnUpdateLocalvideoPlay(CCmdUI* pCmdUI)
{
    int t_Index;

```

```

        if ( !GetIndexFromLabel(t_Index) )
            return;

        pCmdUI->Enable(
!gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
    }

void CLocalVideoEventsView::OnUpdateLocalvideoStop(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
}

void CLocalVideoEventsView::OnLocalvideoMotiondetection()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    t_LocalVideoDoc->UpdateAllViews(NULL,
NUPDATE_LOCALVIDEOMOTIONDETECTION, NULL);
}

bool CLocalVideoEventsView::ReceiveEventsArray(CConnectionMsg*
pConnectionMsg)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL )
        return false;
    gm_Connections.ReceiveEvents(t_LocalVideoDoc->m_ConnectionLabel,
pConnectionMsg);
    UpdateListCtrl();
    return true;
}

void CLocalVideoEventsView::OnUpdateConnectionConnect(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
!gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CLocalVideoEventsView::OnUpdateConnectionDisconnect(CCmdUI*
pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;

```

```

        if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
            pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
    }

void CLocalVideoEventsView::OnDestroy()
{
    CListView::OnDestroy();

    CString t_strConnectionLabel;
    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;
    OnConnectionDisconnect();
}

bool CLocalVideoEventsView::SendEventsArray()
{
    CString t_strConnectionLabel;

    // Make sure there is a connection label
    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return false;

    // Make sure there is a connection!
    if (
!gm_Connections.IsConnectionEstablished(t_strConnectionLabel) )
        return false;

    return gm_Connections.SendEvents(t_strConnectionLabel);
}

void CLocalVideoEventsView::OnUpdateConnectionDeleteEvent(CCmdUI*
pCmdUI)
{
    pCmdUI->Enable( GetSelectedItem() != -1 );
}

bool CLocalVideoEventsView::GetIndexFromLabel(int& t_Index)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();

    if ( t_LocalVideoDoc == NULL )
        return false;

    t_Index = gm_Connections.GetIndexFromLabel(t_LocalVideoDoc-
>m_ConnectionLabel);

    if ( t_Index == -1 )
        return false;

    return true;
}

/*

```

```

void CLocalVideoEventsView::OnUpdateLocalvideoMotiondetection(CCmdUI*
pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
}
*/
// LocalVideoFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "LocalVideoFrame.h"
#include "SimpleVideo.h"
#include "LocalVideoVideoView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CLocalVideoFrame

IMPLEMENT_DYNCREATE(CLocalVideoFrame, CMDIChildWnd)

CLocalVideoFrame::CLocalVideoFrame()
{
}

CLocalVideoFrame::~CLocalVideoFrame()
{
}

BEGIN_MESSAGE_MAP(CLocalVideoFrame, CMDIChildWnd)
    //{AFX_MSG_MAP(CLocalVideoFrame)
    ON_WM_CREATE()
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CLocalVideoFrame message handlers

BOOL CLocalVideoFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    return CMDIChildWnd::PreCreateWindow(cs);
}

```

```

int CLocalVideoFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_LOCALVIDEOSURVEILLANCE))
    {
        return FALSE;        // fail to create
    }

    return 0;
}

BOOL CLocalVideoFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // create a splitter with 2 rows, 1 column
    if (!m_wndSplitter.CreateStatic(this, 1, 2))
    {
        TRACE0("Failed to CreateStaticSplitter\n");
        return FALSE;
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
        pContext->m_pNewViewClass, CSize(130, 50), pContext))
    {
        TRACE0("Failed to create first pane\n");
        return FALSE;
    }

    // add the second splitter pane - an input view in row 0
    if (!m_wndSplitter.CreateView(0, 1,
        RUNTIME_CLASS(CLocalVideoVideoView), CSize(0, 0),
pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(0,0));

    return TRUE;
}

// LocalVideoVideoView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "SimpleVideo.h"
#include "LocalVideoVideoView.h"
#include "LocalVideoDoc.h"

```

```

#include "DirectPlay.h"
#include "ItemListDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CLocalVideoVideoView

IMPLEMENT_DYNCREATE(CLocalVideoVideoView, CView)

CLocalVideoVideoView::CLocalVideoVideoView()
{
}

CLocalVideoVideoView::~CLocalVideoVideoView()
{
}

BEGIN_MESSAGE_MAP(CLocalVideoVideoView, CView)
    //{{AFX_MSG_MAP(CLocalVideoVideoView)
    ON_COMMAND(ID_LOCALVIDEO_PLAY, OnLocalvideoPlay)
    ON_COMMAND(ID_LOCALVIDEO_STOP, OnLocalvideoStop)
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
    ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLargeicons)
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
    ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallicons)
    ON_COMMAND(ID_CONNECTION_CONNECT, OnConnectionConnect)
    ON_COMMAND(ID_CONNECTION_DISCONNECT, OnConnectionDisconnect)
    ON_COMMAND(ID_CONNECTION_NEW_ALARM, OnConnectionNewAlarm)
    ON_COMMAND(ID_CONNECTION_NEW_EVENT, OnConnectionNewEvent)
    ON_COMMAND(ID_CONNECTION_DELETE_EVENT, OnConnectionDeleteEvent)
    ON_COMMAND(ID_LOCALVIDEO_PAUSE, OnLocalvideoPause)
    ON_WM_CONTEXTMENU()
    ON_WM_MOVE()
    ON_WM_SIZE()
    ON_COMMAND(ID_LOCALVIDEO_RECORD, OnLocalvideoRecord)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_PAUSE,
OnUpdateLocalvideoPause)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_PLAY, OnUpdateLocalvideoPlay)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_RECORD,
OnUpdateLocalvideoRecord)
    ON_UPDATE_COMMAND_UI(ID_LOCALVIDEO_STOP, OnUpdateLocalvideoStop)
    ON_COMMAND(ID_LOCALVIDEO_MOTIONDETECTION,
OnLocalvideoMotiondetection)
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_CONNECT,
OnUpdateConnectionConnect)
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_DISCONNECT,
OnUpdateConnectionDisconnect)
    //}}

```

```

        ON_WM_DESTROY()
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_GRAPHNOTIFY, OnGraphNotify)
    END_MESSAGE_MAP()

    //////////////////////////////////////
    // CLocalVideoVideoView drawing

void CLocalVideoVideoView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

    //////////////////////////////////////
    // CLocalVideoVideoView diagnostics

#ifdef _DEBUG
void CLocalVideoVideoView::AssertValid() const
{
    CView::AssertValid();
}

void CLocalVideoVideoView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
#endif // _DEBUG

    //////////////////////////////////////
    // CLocalVideoVideoView message handlers

bool CLocalVideoVideoView::GetIndexFromLabel(int& t_Index)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
    GetDocument();

    if ( t_LocalVideoDoc == NULL )
        return false;

    t_Index = gm_Connections.GetIndexFromLabel(t_LocalVideoDoc->m_ConnectionLabel);

    if ( t_Index == -1 )
        return false;

    return true;
}

void CLocalVideoVideoView::OnLocalvideoPlay()
{
    int t_Index;

```

```

        if ( !GetIndexFromLabel(t_Index) )
            return;

        gm_Connections.ElementAt(t_Index).m_SimpleVideo.Play(m_hWnd);
    }

void CLocalVideoVideoView::OnLocalvideoStop()
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.Disconnect();
}

void CLocalVideoVideoView::OnViewDetails()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_VIEWDETAILS, NULL);
}

void CLocalVideoVideoView::OnViewLargeicons()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_VIEWLARGEICONS, NULL);
}

void CLocalVideoVideoView::OnViewList()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_VIEWLIST, NULL);
}

void CLocalVideoVideoView::OnViewSmallicons()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_VIEWSMALLICONS, NULL);
}

void CLocalVideoVideoView::OnConnectionConnect()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_CONNECT, NULL);
}

void CLocalVideoVideoView::OnConnectionDisconnect()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_DISCONNECT, NULL);
}

```

```

}

void CLocalVideoVideoView::OnConnectionNewAlarm()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUUPDATE_NEWALARM, NULL);
}

void CLocalVideoVideoView::OnConnectionNewEvent()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUUPDATE_NEWSCHEDULEDEVENT,
NULL);
}

void CLocalVideoVideoView::OnConnectionDeleteEvent()
{
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUUPDATE_DELETEEVENT, NULL);
}

void CLocalVideoVideoView::OnLocalvideoPause()
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.VideoPlaybackTogglePause();
}

void CLocalVideoVideoView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    DisplayContextMenu(this, point, IDR_POPUP_LOCALVIDEO);
}

void CLocalVideoVideoView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
    case NUUPDATE_LOCALVIDEOPAUSE:
        OnLocalvideoPause() ;
        break;
    case NUUPDATE_LOCALVIDEOPLAY:
        OnLocalvideoPlay() ;
        break;
    case NUUPDATE_LOCALVIDEOSTOP:
        OnLocalvideoStop() ;
        break;
    case NUUPDATE_LOCALVIDEORECORD:
        OnLocalvideoRecord() ;
        break;
    }
}

```

```

        case NUUPDATE_LOCALVIDEOMOTIONDETECTION:
            OnLocalvideoMotiondetection();
            break;
    };
}

void CLocalVideoVideoView::OnMove(int x, int y)
{
    CView::OnMove(x, y);
    CRect rect;
    GetClientRect(&rect);
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.VideoWindowSetWin
dowPos(rect);
}

void CLocalVideoVideoView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    CRect rect;
    GetClientRect(&rect);
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.VideoWindowSetWin
dowPos(rect);
}

void CLocalVideoVideoView::OnLocalvideoRecord()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    if ( t_LocalVideoDoc == NULL )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.SetRecordTrigger(
RECORDTRIGGER_USER);
    gm_Connections.ElementAt(t_Index).m_SimpleVideo.ToggleRecord(m_hW
nd, t_LocalVideoDoc->m_ConnectionLabel);
}

void CLocalVideoVideoView::OnUpdateLocalvideoPause(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )

```

```

        return;

        pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_PLAYING );
    }

void CLocalVideoVideoView::OnUpdateLocalvideoPlay(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
!gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
}

void CLocalVideoVideoView::OnUpdateLocalvideoRecord(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->SetCheck(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_RECORDING ||

gm_Connections.ElementAt(t_Index).m_SimpleVideo.m_State ==
SIMPLEVIDEOSTATE_STREAMINGANDRECORDING );
}

void CLocalVideoVideoView::OnUpdateLocalvideoStop(CCmdUI* pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
}

LRESULT CLocalVideoVideoView::OnGraphNotify( WPARAM wParam, LPARAM
lParam )
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        goto Exit1;

    if ( t_LocalVideoDoc == NULL )
        goto Exit1;
}

```

```

        gm_Connections.ElementAt(t_Index).m_SimpleVideo.HandleGraphEvent(
t_LocalVideoDoc->m_ConnectionLabel);

Exit1:
    return S_OK;
}

void CLocalVideoVideoView::OnLocalvideoMotiondetection()
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.PromptMotionDetec
tionFilterPropPage(m_hWnd);
    //
    gm_Connections.ElementAt(t_Index).m_SimpleVideo.PromptMotionDetec
tionDlg(m_hWnd);
}

void CLocalVideoVideoView::OnUpdateConnectionConnect(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
!gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CLocalVideoVideoView::OnUpdateConnectionDisconnect(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CLocalVideoVideoView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;

    t_LocalVideoDoc->m_hWndVideoView = m_hWnd;
    m_nTimer = SetTimer(11, 600000, NULL);
}

void CLocalVideoVideoView::OnDestroy()

```

```

{
    CView::OnDestroy();

    if ( m_nTimer != NULL )
        KillTimer( m_nTimer );
}

void CLocalVideoVideoView::OnTimer(UINT nIDEvent)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    gm_Connections.ElementAt(t_Index).m_SimpleVideo.InterruptStream();
;

    CView::OnTimer(nIDEvent);
}

/*
void CLocalVideoVideoView::OnUpdateLocalvideoMotiondetection(CCmdUI*
pCmdUI)
{
    int t_Index;

    if ( !GetIndexFromLabel(t_Index) )
        return;

    pCmdUI->Enable(
gm_Connections.ElementAt(t_Index).m_SimpleVideo.IsActive() );
}
*/
// Main Doc.cpp : implementation of the CMainDoc class
//

#include "stdafx.h"
#include "Project Nayal.h"

#include "Main Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CMainDoc

IMPLEMENT_DYNCREATE(CMainDoc, CDocument)

BEGIN_MESSAGE_MAP(CMainDoc, CDocument)
//{{AFX_MSG_MAP(CMainDoc)
// NOTE - the ClassWizard will add and remove mapping
macros here.

```

```

        // DO NOT EDIT what you see in these blocks of generated
code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
/////
// CMainDoc construction/destruction

```

```

CMainDoc::CMainDoc()
{
    // TODO: add one-time construction code here
}

```

```

CMainDoc::~CMainDoc()
{
}

```

```

BOOL CMainDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}

```

```

////////////////////////////////////
/////
// CMainDoc serialization

```

```

void CMainDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

```

```

////////////////////////////////////
/////
// CMainDoc diagnostics

```

```

#ifdef _DEBUG
void CMainDoc::AssertValid() const
{
    CDocument::AssertValid();
}

```

```

}

void CMainDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////
////////
// CMainDoc commands

// Main View.cpp : implementation of the CMainView class
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "Main Doc.h"
#include "Main View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
////////
// CMainView

IMPLEMENT_DYNCREATE(CMainView, CView)

BEGIN_MESSAGE_MAP(CMainView, CView)
    //{AFX_MSG_MAP(CMainView)
    // NOTE - the ClassWizard will add and remove mapping
macros here.
    // DO NOT EDIT what you see in these blocks of generated
code!
    //{AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////
////////
// CMainView construction/destruction

CMainView::CMainView()
{
    // TODO: add construction code here
}

```

```

CMainView::~CMainView()
{
}

BOOL CMainView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

////////////////////////////////////
/////
// CMainView drawing

void CMainView::OnDraw(CDC* pDC)
{
    CMainDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
}

////////////////////////////////////
/////
// CMainView printing

BOOL CMainView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMainView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMainView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

////////////////////////////////////
/////
// CMainView diagnostics

#ifdef _DEBUG
void CMainView::AssertValid() const
{
    CView::AssertValid();
}

void CMainView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

```

DEMANDES OU BREVETS VOLUMINEUX

**LA PRÉSENTE PARTIE DE CETTE DEMANDE OU CE BREVETS
COMPREND PLUS D'UN TOME.**

CECI EST LE TOME 1 DE 2

NOTE: Pour les tomes additionels, veuillez contacter le Bureau Canadien des Brevets.

JUMBO APPLICATIONS / PATENTS

**THIS SECTION OF THE APPLICATION / PATENT CONTAINS MORE
THAN ONE VOLUME.**

THIS IS VOLUME 1 OF 2

NOTE: For additional volumes please contact the Canadian Patent Office.

DEMANDES OU BREVETS VOLUMINEUX

**LA PRÉSENTE PARTIE DE CETTE DEMANDE OU CE BREVETS
COMPREND PLUS D'UN TOME.**

CECI EST LE TOME 2 DE 2

NOTE: Pour les tomes additionels, veuillez contacter le Bureau Canadien des Brevets.

JUMBO APPLICATIONS / PATENTS

**THIS SECTION OF THE APPLICATION / PATENT CONTAINS MORE
THAN ONE VOLUME.**

THIS IS VOLUME 2 OF 2

NOTE: For additional volumes please contact the Canadian Patent Office.

```

CMainDoc* CMainView::GetDocument() // non-debug version is inline

{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMainDoc)));
    return (CMainDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
/////
// CMainView message handlers
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;
extern CConnectionsArray gm_Connections;

////////////////////////////////////
/////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_TIMER()
    ON_WM_CLOSE()
    //}}AFX_MSG_MAP
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,

```

```

        ID_INDICATOR_SCROLL,
    };

    //////////////////////////////////////
    //////////////////////////////////////
    // CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;    // fail to create
    }

    /* LCK - Remove dialog bar
    if (!m_wndDlgBar.Create(this, IDR_MAINFRAME,
        CBRS_ALIGN_TOP, AFX_IDW_DIALOGBAR))
    {
        TRACE0("Failed to create dialogbar\n");
        return -1;    // fail to create
    }
    */

    if (!m_wndReBar.Create(this) ||
        !m_wndReBar.AddBar(&m_wndToolBar) ||
        // !m_wndReBar.AddBar(&m_wndToolBar) ||
        // !m_wndReBar.AddBar(&m_wndDlgBar))
    {
        TRACE0("Failed to create rebar\n");
        return -1;    // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;    // fail to create
    }

    // TODO: Remove this if you don't want tool tips
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY);

```

```

        m_nTimer = SetTimer(10, 60000, NULL);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CMDIFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Modify the Window class or styles here by modifying
        // the CREATESTRUCT cs

        cs.x = gm_Layout.m_rectMainWindow.left;
        cs.y = gm_Layout.m_rectMainWindow.top;
        cs.cx = gm_Layout.m_rectMainWindow.Width();
        cs.cy = gm_Layout.m_rectMainWindow.Height();

        cs.style |= WS_VSCROLL | WS_HSCROLL;

        return TRUE;
    }

    //////////////////////////////////////
    //////////////////////////////////////
    // CMainFrame diagnostics

    #ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CMDIFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CMDIFrameWnd::Dump(dc);
    }

    #endif // _DEBUG

    //////////////////////////////////////
    //////////////////////////////////////
    // CMainFrame message handlers

    void CMainFrame::OnDestroy()
    {
        CMDIFrameWnd::OnDestroy();

        if ( m_nTimer != NULL )
            KillTimer( m_nTimer );

        GetWindowRect(&gm_Layout.m_rectMainWindow);
    }

```

```

void CMainFrame::OnTimer(UINT nIDEvent)
{
    CString t_strDescription;
    CTime t_Time;
    int t_CurHour, t_CurMinute;
    t_Time = CTime::GetCurrentTime();
    t_CurHour = t_Time.GetHour();
    t_CurMinute = t_Time.GetMinute();
    t_strDescription.Format("Hour:Minute: %02d:%02d", t_CurHour,
t_CurMinute);
    // t_str = t_Time.Format("%D:%H:%M:%S");
    CString t_strMessage;
    t_strMessage.LoadString(IDS_TIMER);

    NSENDFEEDBACKMESSAGE(FEEDBACKMESSAGE_TYPE_DEBUG, t_strMessage,
t_strDescription);

    CTime t_CurrentTime;
    t_CurrentTime = CTime::GetCurrentTime();

    gm_Connections.TriggerScheduledEvent(t_CurrentTime);

    CMDIFrameWnd::OnTimer(nIDEvent);
}

void CMainFrame::OnClose()
{
    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONEXIT) )
        return ;

    CMDIFrameWnd::OnClose();
}
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

#include "stdafx.h"
#include "mediaplayer2.h"

// Dispatch interfaces referenced by this interface
#include "MediaPlayerDvd.h"

////////////////////////////////////
/////
// CMediaPlayer2

IMPLEMENT_DYNCREATE(CMediaPlayer2, CWnd)

////////////////////////////////////
/////

```

```

// CMediaPlayer2 properties

////////////////////////////////////
////////
// CMediaPlayer2 operations

double CMediaPlayer2::GetCurrentPosition()
{
    double result;
    InvokeHelper(0x403, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetCurrentPosition(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x403, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

double CMediaPlayer2::GetDuration()
{
    double result;
    InvokeHelper(0x3eb, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
    NULL);
    return result;
}

long CMediaPlayer2::GetImageSourceWidth()
{
    long result;
    InvokeHelper(0x3e9, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

long CMediaPlayer2::GetImageSourceHeight()
{
    long result;
    InvokeHelper(0x3ea, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

long CMediaPlayer2::GetMarkerCount()
{
    long result;
    InvokeHelper(0x3f2, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

BOOL CMediaPlayer2::GetCanScan()
{
    BOOL result;

```

```

        InvokeHelper(0x3f3, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

    BOOL CMediaPlayer2::GetCanSeek()
    {
        BOOL result;
        InvokeHelper(0x3f4, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

    BOOL CMediaPlayer2::GetCanSeekToMarkers()
    {
        BOOL result;
        InvokeHelper(0x417, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

    long CMediaPlayer2::GetCurrentMarker()
    {
        long result;
        InvokeHelper(0x405, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
        return result;
    }

    void CMediaPlayer2::SetCurrentMarker(long nNewValue)
    {
        static BYTE parms[] =
            VTS_I4;
        InvokeHelper(0x405, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
nNewValue);
    }

    CString CMediaPlayer2::GetFileName()
    {
        CString result;
        InvokeHelper(0x402, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetFileName(LPCTSTR lpszNewValue)
    {
        static BYTE parms[] =
            VTS_BSTR;
        InvokeHelper(0x402, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
lpszNewValue);
    }

    CString CMediaPlayer2::GetSourceLink()
    {
        CString result;

```

```
        InvokeHelper(0x3f1, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
        return result;
    }

DATE CMediaPlayer2::GetCreationDate()
{
    DATE result;
    InvokeHelper(0x40c, DISPATCH_PROPERTYGET, VT_DATE,
(void*)&result, NULL);
    return result;
}

CString CMediaPlayer2::GetErrorCorrection()
{
    CString result;
    InvokeHelper(0x40e, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

long CMediaPlayer2::GetBandwidth()
{
    long result;
    InvokeHelper(0x40d, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CMediaPlayer2::GetSourceProtocol()
{
    long result;
    InvokeHelper(0x424, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CMediaPlayer2::GetReceivedPackets()
{
    long result;
    InvokeHelper(0x40f, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CMediaPlayer2::GetRecoveredPackets()
{
    long result;
    InvokeHelper(0x410, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CMediaPlayer2::GetLostPackets()
{
    long result;
```

```

        InvokeHelper(0x411, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
        return result;
    }

long CMediaPlayer2::GetReceptionQuality()
{
    long result;
    InvokeHelper(0x412, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

long CMediaPlayer2::GetBufferingCount()
{
    long result;
    InvokeHelper(0x413, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

BOOL CMediaPlayer2::GetIsBroadcast()
{
    BOOL result;
    InvokeHelper(0x422, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

long CMediaPlayer2::GetBufferingProgress()
{
    long result;
    InvokeHelper(0x438, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

CString CMediaPlayer2::GetChannelName()
{
    CString result;
    InvokeHelper(0x41a, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

CString CMediaPlayer2::GetChannelDescription()
{
    CString result;
    InvokeHelper(0x41b, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

CString CMediaPlayer2::GetChannelURL()
{
    CString result;

```

```

        InvokeHelper(0x41c, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
        return result;
    }

CString CMediaPlayer2::GetContactAddress()
{
    CString result;
    InvokeHelper(0x41d, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

CString CMediaPlayer2::GetContactPhone()
{
    CString result;
    InvokeHelper(0x41e, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

CString CMediaPlayer2::GetContactEmail()
{
    CString result;
    InvokeHelper(0x41f, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

double CMediaPlayer2::GetBufferingTime()
{
    double result;
    InvokeHelper(0x42e, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
NULL);
    return result;
}

void CMediaPlayer2::SetBufferingTime(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x42e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

BOOL CMediaPlayer2::GetAutoStart()
{
    BOOL result;
    InvokeHelper(0x3f9, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);

    return result;
}

void CMediaPlayer2::SetAutoStart(BOOL bNewValue)
{
    static BYTE parms[] =

```

```

        VTS_BOOL;
        InvokeHelper(0x3f9, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetAutoRewind()
    {
        BOOL result;
        InvokeHelper(0x3fa, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetAutoRewind(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x3fa, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    double CMediaPlayer2::GetRate()
    {
        double result;
        InvokeHelper(0x404, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
            NULL);
        return result;
    }

    void CMediaPlayer2::SetRate(double newValue)
    {
        static BYTE parms[] =
            VTS_R8;
        InvokeHelper(0x404, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            newValue);
    }

    BOOL CMediaPlayer2::GetSendKeyboardEvents()
    {
        BOOL result;
        InvokeHelper(0x3f5, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetSendKeyboardEvents(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x3f5, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetSendMouseClickEvents()
    {
        BOOL result;

```

```

    InvokeHelper(0x3f6, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSendMouseClickEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x3f6, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetSendMouseMoveEvents()
{
    BOOL result;
    InvokeHelper(0x3f7, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSendMouseMoveEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x3f7, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CMediaPlayer2::GetPlayCount()
{
    long result;
    InvokeHelper(0x406, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CMediaPlayer2::SetPlayCount(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x406, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CMediaPlayer2::GetClickToPlay()
{
    BOOL result;
    InvokeHelper(0x401, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetClickToPlay(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;

```

```

        InvokeHelper(0x401, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetAllowScan()
    {
        BOOL result;
        InvokeHelper(0x40b, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetAllowScan(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x40b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetEnableContextMenu()
    {
        BOOL result;
        InvokeHelper(0x3fd, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetEnableContextMenu(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x3fd, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    long CMediaPlayer2::GetCursorType()
    {
        long result;
        InvokeHelper(0x414, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
            NULL);
        return result;
    }

    void CMediaPlayer2::SetCursorType(long nNewValue)
    {
        static BYTE parms[] =
            VTS_I4;
        InvokeHelper(0x414, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            nNewValue);
    }

    long CMediaPlayer2::GetCodecCount()
    {
        long result;

```

```

    InvokeHelper(0x421, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

BOOL CMediaPlayer2::GetAllowChangeDisplaySize()
{
    BOOL result;
    InvokeHelper(0x420, DISPATCH_PROPERTYGET, VT_BOOL,
    (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetAllowChangeDisplaySize(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x420, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetIsDurationValid()
{
    BOOL result;
    InvokeHelper(0x423, DISPATCH_PROPERTYGET, VT_BOOL,
    (void*)&result, NULL);
    return result;
}

long CMediaPlayer2::GetOpenState()
{
    long result;
    InvokeHelper(0x425, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

BOOL CMediaPlayer2::GetSendOpenStateChangeEvents()
{
    BOOL result;
    InvokeHelper(0x426, DISPATCH_PROPERTYGET, VT_BOOL,
    (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSendOpenStateChangeEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x426, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetSendWarningEvents()
{
    BOOL result;

```

```

        InvokeHelper(0x427, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

void CMediaPlayer2::SetSendWarningEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x427, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetSendErrorEvents()
{
    BOOL result;
    InvokeHelper(0x428, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSendErrorEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x428, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CMediaPlayer2::GetPlayState()
{
    long result;
    InvokeHelper(0x42c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

BOOL CMediaPlayer2::GetSendPlayStateChangeEvents()
{
    BOOL result;
    InvokeHelper(0x42d, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSendPlayStateChangeEvents(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x42d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CMediaPlayer2::GetDisplaySize()
{
    long result;

```

```

        InvokeHelper(0x408, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
        return result;
    }

void CMediaPlayer2::SetDisplaySize(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x408, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CMediaPlayer2::GetInvokeURLs()
{
    BOOL result;
    InvokeHelper(0x3fc, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetInvokeURLs(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x3fc, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

CString CMediaPlayer2::GetBaseURL()
{
    CString result;
    InvokeHelper(0x43a, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetBaseURL(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x43a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

CString CMediaPlayer2::GetDefaultFrame()
{
    CString result;
    InvokeHelper(0x43b, DISPATCH_PROPERTYGET, VT_BSTR,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetDefaultFrame(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;

```

```

        InvokeHelper(0x43b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            lpszNewValue);
    }

    BOOL CMediaPlayer2::GetHasError()
    {
        BOOL result;
        InvokeHelper(0x429, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    CString CMediaPlayer2::GetErrorDescription()
    {
        CString result;
        InvokeHelper(0x42a, DISPATCH_PROPERTYGET, VT_BSTR,
            (void*)&result, NULL);
        return result;
    }

    long CMediaPlayer2::GetErrorCode()
    {
        long result;
        InvokeHelper(0x42b, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
            NULL);
        return result;
    }

    BOOL CMediaPlayer2::GetAnimationAtStart()
    {
        BOOL result;
        InvokeHelper(0x415, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetAnimationAtStart(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x415, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetTransparentAtStart()
    {
        BOOL result;
        InvokeHelper(0x3fe, DISPATCH_PROPERTYGET, VT_BOOL,
            (void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetTransparentAtStart(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x3fe, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,

```

```

        bNewValue);
    }

long CMediaPlayer2::GetVolume()
{
    long result;
    InvokeHelper(0x13, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetVolume(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x13, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

long CMediaPlayer2::GetBalance()
{
    long result;
    InvokeHelper(0x14, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetBalance(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x14, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

long CMediaPlayer2::GetReadyState()
{
    long result;
    InvokeHelper(DISPID_READYSTATE, DISPATCH_PROPERTYGET, VT_I4,
    (void*)&result, NULL);
    return result;
}

double CMediaPlayer2::GetSelectionStart()
{
    double result;
    InvokeHelper(0xf, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetSelectionStart(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0xf, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,

```

```

        newValue);
    }

double CMediaPlayer2::GetSelectionEnd()
{
    double result;
    InvokeHelper(0x10, DISPATCH_PROPERTYGET, VT_R8, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetSelectionEnd(double newValue)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x10, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

BOOL CMediaPlayer2::GetShowDisplay()
{
    BOOL result;
    InvokeHelper(0x16, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetShowDisplay(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x16, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowControls()
{
    BOOL result;
    InvokeHelper(0x17, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayer2::SetShowControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x17, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowPositionControls()
{
    BOOL result;
    InvokeHelper(0x18, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
    NULL);
    return result;
}

```

```

}

void CMediaPlayer2::SetShowPositionControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x18, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowTracker()
{
    BOOL result;
    InvokeHelper(0x1a, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetShowTracker(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x1a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetEnablePositionControls()
{
    BOOL result;
    InvokeHelper(0x1b, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetEnablePositionControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x1b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetEnableTracker()
{
    BOOL result;
    InvokeHelper(0x1d, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetEnableTracker(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x1d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

```

```

}

BOOL CMediaPlayer2::GetEnabled()
{
    BOOL result;
    InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetEnabled(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYPUT, VT_EMPTY,
NULL, parms,
        bNewValue);
}

unsigned long CMediaPlayer2::GetDisplayForeColor()
{
    unsigned long result;
    InvokeHelper(0x24, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CMediaPlayer2::SetDisplayForeColor(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x24, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

unsigned long CMediaPlayer2::GetDisplayBackColor()
{
    unsigned long result;
    InvokeHelper(0x25, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

void CMediaPlayer2::SetDisplayBackColor(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x25, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

long CMediaPlayer2::GetDisplayMode()
{
    long result;
    InvokeHelper(0x20, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
    return result;
}

```

```

}

void CMediaPlayer2::SetDisplayMode(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x20, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

BOOL CMediaPlayer2::GetVideoBorder3D()
{
    BOOL result;
    InvokeHelper(0x44f, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetVideoBorder3D(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x44f, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

long CMediaPlayer2::GetVideoBorderWidth()
{
    long result;
    InvokeHelper(0x44d, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetVideoBorderWidth(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x44d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

unsigned long CMediaPlayer2::GetVideoBorderColor()
{
    unsigned long result;
    InvokeHelper(0x44e, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetVideoBorderColor(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x44e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

```

```

BOOL CMediaPlayer2::GetShowGotoBar()
{
    BOOL result;
    InvokeHelper(0x440, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetShowGotoBar(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x440, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowStatusBar()
{
    BOOL result;
    InvokeHelper(0x43e, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetShowStatusBar(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x43e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowCaptioning()
{
    BOOL result;
    InvokeHelper(0x43c, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetShowCaptioning(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x43c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetShowAudioControls()
{
    BOOL result;
    InvokeHelper(0x453, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
    return result;
}

```

```

void CMediaPlayer2::SetShowAudioControls(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x453, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

CString CMediaPlayer2::GetCaptioningID()
{
    CString result;
    InvokeHelper(0x43d, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetCaptioningID(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x43d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

BOOL CMediaPlayer2::GetMute()
{
    BOOL result;
    InvokeHelper(0x441, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetMute(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x441, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetCanPreview()
{
    BOOL result;
    InvokeHelper(0x445, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

BOOL CMediaPlayer2::GetPreviewMode()
{
    BOOL result;
    InvokeHelper(0x443, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetPreviewMode(BOOL bNewValue)

```

```

{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x443, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

BOOL CMediaPlayer2::GetHasMultipleItems()
{
    BOOL result;
    InvokeHelper(0x446, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

long CMediaPlayer2::GetLanguage()
{
    long result;
    InvokeHelper(0x447, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetLanguage(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x447, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

long CMediaPlayer2::GetAudioStream()
{
    long result;
    InvokeHelper(0x448, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::SetAudioStream(long nNewValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x448, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        nNewValue);
}

CString CMediaPlayer2::GetSAMISStyle()
{
    CString result;
    InvokeHelper(0x449, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSAMISStyle(LPCTSTR lpszNewValue)
{

```

```

        static BYTE parms[] =
            VTS_BSTR;
        InvokeHelper(0x449, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            lpszNewValue);
    }

CString CMediaPlayer2::GetSAMILang()
{
    CString result;
    InvokeHelper(0x44a, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSAMILang(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x44a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

CString CMediaPlayer2::GetSAMIFileName()
{
    CString result;
    InvokeHelper(0x44b, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayer2::SetSAMIFileName(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x44b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

long CMediaPlayer2::GetStreamCount()
{
    long result;
    InvokeHelper(0x44c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

CString CMediaPlayer2::GetClientId()
{
    CString result;
    InvokeHelper(0x452, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

long CMediaPlayer2::GetConnectionSpeed()
{
    long result;

```

```

        Invok Helper(0x459, DISPATCH_PROPERTYGET, VT_I4, (void*)&r sult,
NULL);
        return result;
    }

    BOOL CMediaPlayer2::GetAutoSize()
    {
        BOOL result;
        InvokeHelper(0xfffffe0c, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetAutoSize(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0xfffffe0c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL,
parms,
            bNewValue);
    }

    BOOL CMediaPlayer2::GetEnableFullScreenControls()
    {
        BOOL result;
        InvokeHelper(0x454, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

    void CMediaPlayer2::SetEnableFullScreenControls(BOOL bNewValue)
    {
        static BYTE parms[] =
            VTS_BOOL;
        InvokeHelper(0x454, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
            bNewValue);
    }

    LPDISPATCH CMediaPlayer2::GetActiveMovie()
    {
        LPDISPATCH result;
        InvokeHelper(0x455, DISPATCH_PROPERTYGET, VT_DISPATCH,
(void*)&result, NULL);
        return result;
    }

    LPDISPATCH CMediaPlayer2::GetNSPlay()
    {
        LPDISPATCH result;
        InvokeHelper(0x456, DISPATCH_PROPERTYGET, VT_DISPATCH,
(void*)&result, NULL);
        return result;
    }

    BOOL CMediaPlayer2::GetWindowlessVideo()
    {
        BOOL result;

```

```

        InvokeHelper(0x458, DISPATCH_PROPERTYGET, VT_BOOL,
(void*)&result, NULL);
        return result;
    }

void CMediaPlayer2::SetWindowlessVideo(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x458, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

void CMediaPlayer2::Play()
{
    InvokeHelper(0x7d1, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayer2::Stop()
{
    InvokeHelper(0x7d3, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayer2::Pause()
{
    InvokeHelper(0x7d2, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

double CMediaPlayer2::GetMarkerTime(long MarkerNum)
{
    double result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7d4, DISPATCH_METHOD, VT_R8, (void*)&result,
parms,
        MarkerNum);
    return result;
}

CString CMediaPlayer2::GetMarkerName(long MarkerNum)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7d5, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        MarkerNum);
    return result;
}

void CMediaPlayer2::AboutBox()
{
    InvokeHelper(0xfffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

BOOL CMediaPlayer2::GetCodecInstalled(long CodecNum)

```

```

{
    BOOL result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7d7, DISPATCH_METHOD, VT_BOOL, (void*)&result,
parms,
        CodecNum);
    return result;
}

CString CMediaPlayer2::GetCodecDescription(long CodecNum)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7d8, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        CodecNum);
    return result;
}

CString CMediaPlayer2::GetCodecURL(long CodecNum)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7d9, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        CodecNum);
    return result;
}

CString CMediaPlayer2::GetMoreInfoURL(long MoreInfoType)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7db, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        MoreInfoType);
    return result;
}

CString CMediaPlayer2::GetMediaInfoString(long MediaInfoType)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7e0, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        MediaInfoType);
    return result;
}

void CMediaPlayer2::Cancel()
{

```

```

        InvokeHelper(0x7d6, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
    }

void CMediaPlayer2::Open(LPCTSTR bstrFileName )
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x7da, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        bstrFileName);
}

BOOL CMediaPlayer2::IsSoundCardEnabled()
{
    BOOL result;
    InvokeHelper(0x35, DISPATCH_METHOD, VT_BOOL, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayer2::Next()
{
    InvokeHelper(0x7e7, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayer2::Previous()
{
    InvokeHelper(0x7e6, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayer2::StreamSelect(long StreamNum)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7df, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        StreamNum);
}

void CMediaPlayer2::FastForward()
{
    InvokeHelper(0x7e8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayer2::FastReverse()
{
    InvokeHelper(0x7e9, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

CString CMediaPlayer2::GetStreamName(long StreamNum)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7e3, DISPATCH_METHOD, VT_BSTR, (void*)&result,
        parms,
        StreamNum);
    return result;
}

```

```

long CMediaPlayer2::GetStreamGroup(long StreamNum)
{
    long result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7e4, DISPATCH_METHOD, VT_I4, (void*)&result,
parms,
        StreamNum);
    return result;
}

BOOL CMediaPlayer2::GetStreamSelected(long StreamNum)
{
    BOOL result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7e5, DISPATCH_METHOD, VT_BOOL, (void*)&result,
parms,
        StreamNum);
    return result;
}

CMediaPlayerDvd CMediaPlayer2::GetDvd()
{
    LPDISPATCH pDispatch;
    InvokeHelper(0x5dc, DISPATCH_PROPERTYGET, VT_DISPATCH,
(void*)&pDispatch, NULL);
    return CMediaPlayerDvd(pDispatch);
}

CString CMediaPlayer2::GetMediaParameter(long EntryNum, LPCTSTR
bstrParameterName)
{
    CString result;
    static BYTE parms[] =
        VTS_I4 VTS_BSTR;
    InvokeHelper(0x7ec, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        EntryNum, bstrParameterName);
    return result;
}

CString CMediaPlayer2::GetMediaParameterName(long EntryNum, long Index)
{
    CString result;
    static BYTE parms[] =
        VTS_I4 VTS_I4;
    InvokeHelper(0x7ed, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        EntryNum, Index);
    return result;
}

long CMediaPlayer2::GetEntryCount()
{
    long result;

```

```

        InvokeHelper(0x7ee, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
NULL);
        return result;
    }

long CMediaPlayer2::GetCurrentEntry()
{
    long result;
    InvokeHelper(0x7ef, DISPATCH_METHOD, VT_I4, (void*)&result,
NULL);
    return result;
}

void CMediaPlayer2::SetCurrentEntry(long EntryNumber)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7f0, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        EntryNumber);
}

void CMediaPlayer2::ShowDialog(long mpDialogIndex)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x7f1, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        mpDialogIndex);
}
// Machine generated IDispatch wrapper class(es) created by Microsoft
Visual C++

// NOTE: Do not modify the contents of this file.  If this class is
regenerated by
// Microsoft Visual C++, your modifications will be overwritten.

#include "stdafx.h"
#include "mediaplayerdvd.h"

////////////////////////////////////
/////
// CMediaPlayerDvd properties

////////////////////////////////////
/////
// CMediaPlayerDvd operations

void CMediaPlayerDvd::ButtonSelectAndActivate(unsigned long uiButton)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x5f6, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        uiButton);
}

void CMediaPlayerDvd::UpperButtonSelect()

```

```

    {
        InvokeHelper(0x5f1, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
    }

void CMediaPlayerDvd::LowerButtonSelect()
{
    InvokeHelper(0x5f2, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::LeftButtonSelect()
{
    InvokeHelper(0x5f3, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::RightButtonSelect()
{
    InvokeHelper(0x5f4, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::ButtonActivate()
{
    InvokeHelper(0x5f5, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::ForwardScan(double dwSpeed)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x5ed, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        dwSpeed);
}

void CMediaPlayerDvd::BackwardScan(double dwSpeed)
{
    static BYTE parms[] =
        VTS_R8;
    InvokeHelper(0x5ee, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        dwSpeed);
}

void CMediaPlayerDvd::PrevPGSearch()
{
    InvokeHelper(0x5ea, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::TopPGSearch()
{
    InvokeHelper(0x5eb, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::NextPGSearch()
{
    InvokeHelper(0x5ec, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::TitlePlay(unsigned long uiTitle)
{

```

```

        static BYTE parms[] =
            VTS_I4;
        InvokeHelper(0x5e3, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
            uiTitle);
    }

void CMediaPlayerDvd::ChapterPlay(unsigned long uiTitle, unsigned long
uiChapter)
{
    static BYTE parms[] =
        VTS_I4 VTS_I4;
    InvokeHelper(0x5e4, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        uiTitle, uiChapter);
}

void CMediaPlayerDvd::ChapterSearch(unsigned long Chapter)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x5e9, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        Chapter);
}

void CMediaPlayerDvd::MenuCall(long MenuID)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x5ef, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        MenuID);
}

void CMediaPlayerDvd::ResumeFromMenu()
{
    InvokeHelper(0x5f0, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::TimePlay(unsigned long uiTitle, LPCTSTR bstrTime)
{
    static BYTE parms[] =
        VTS_I4 VTS_BSTR;
    InvokeHelper(0x5e5, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        uiTitle, bstrTime);
}

void CMediaPlayerDvd::TimeSearch(LPCTSTR bstrTime)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x5e8, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
        bstrTime);
}

void CMediaPlayerDvd::ChapterPlayAutoStop(unsigned long ulTitle,
unsigned long ulChapter, unsigned long ulChaptersToPlay)
{
    static BYTE parms[] =
        VTS_I4 VTS_I4 VTS_I4;

```

```

        InvokeHelper(0x605, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
            ulTitle, ulChapter, ulChaptersToPlay);
    }

void CMediaPlayerDvd::StillOff()
{
    InvokeHelper(0x5f7, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CMediaPlayerDvd::GoUp()
{
    InvokeHelper(0x5e7, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

CString CMediaPlayerDvd::GetTotalTitleTime()
{
    CString result;
    InvokeHelper(0x62e, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetNumberOfChapters(unsigned long
ulTitle)
{
    unsigned long result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x60e, DISPATCH_METHOD, VT_I4, (void*)&result,
parms,
        ulTitle);
    return result;
}

CString CMediaPlayerDvd::GetAudioLanguage(unsigned long ulStream)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x60f, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        ulStream);
    return result;
}

CString CMediaPlayerDvd::GetSubpictureLanguage(unsigned long ulStream)
{
    CString result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x613, DISPATCH_METHOD, VT_BSTR, (void*)&result,
parms,
        ulStream);
    return result;
}

VARIANT CMediaPlayerDvd::GetAllGPRMs()

```

```

{
    VARIANT result;
    InvokeHelper(0x618, DISPATCH_METHOD, VT_VARIANT, (void*)&result,
    NULL);
    return result;
}

VARIANT CMediaPlayerDvd::GetAllSPRMs()
{
    VARIANT result;
    InvokeHelper(0x617, DISPATCH_METHOD, VT_VARIANT, (void*)&result,
    NULL);
    return result;
}

BOOL CMediaPlayerDvd::UOPValid(unsigned long ulUOP)
{
    BOOL result;
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x62b, DISPATCH_METHOD, VT_BOOL, (void*)&result,
    parms,
        ulUOP);
    return result;
}

unsigned long CMediaPlayerDvd::GetButtonsAvailable()
{
    unsigned long result;
    InvokeHelper(0x623, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentButton()
{
    unsigned long result;
    InvokeHelper(0x622, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetAudioStreamsAvailable()
{
    unsigned long result;
    InvokeHelper(0x607, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentAudioStream()
{
    unsigned long result;
    InvokeHelper(0x608, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

```

```

void CMediaPlayerDvd::SetCurrentAudioStream(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x608, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

unsigned long CMediaPlayerDvd::GetCurrentSubpictureStream()
{
    unsigned long result;
    InvokeHelper(0x609, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayerDvd::SetCurrentSubpictureStream(unsigned long
newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x609, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

unsigned long CMediaPlayerDvd::GetSubpictureStreamsAvailable()
{
    unsigned long result;
    InvokeHelper(0x60a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

BOOL CMediaPlayerDvd::GetSubpictureOn()
{
    BOOL result;
    InvokeHelper(0x60b, DISPATCH_PROPERTYGET, VT_BOOL,
        (void*)&result, NULL);
    return result;
}

void CMediaPlayerDvd::SetSubpictureOn(BOOL bNewValue)
{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x60b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

unsigned long CMediaPlayerDvd::GetAnglesAvailable()
{
    unsigned long result;
    InvokeHelper(0x60d, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

```

```

unsigned long CMediaPlayerDvd::GetCurrentAngle()
{
    unsigned long result;
    InvokeHelper(0x60c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

void CMediaPlayerDvd::SetCurrentAngle(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x60c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

unsigned long CMediaPlayerDvd::GetCurrentTitle()
{
    unsigned long result;
    InvokeHelper(0x61f, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentChapter()
{
    unsigned long result;
    InvokeHelper(0x620, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

CString CMediaPlayerDvd::GetCurrentTime()
{
    CString result;
    InvokeHelper(0x621, DISPATCH_PROPERTYGET, VT_BSTR,
    (void*)&result, NULL);
    return result;
}

void CMediaPlayerDvd::SetRoot(LPCTSTR lpszNewValue)
{
    static BYTE parms[] =
        VTS_BSTR;
    InvokeHelper(0x602, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        lpszNewValue);
}

CString CMediaPlayerDvd::GetRoot()
{
    CString result;
    InvokeHelper(0x602, DISPATCH_PROPERTYGET, VT_BSTR,
    (void*)&result, NULL);
    return result;
}

```

```
unsigned long CMediaPlayerDvd::GetFramesPerSecond()
{
    unsigned long result;
    InvokeHelper(0x625, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentDomain()
{
    unsigned long result;
    InvokeHelper(0x626, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetTitlesAvailable()
{
    unsigned long result;
    InvokeHelper(0x627, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetVolumesAvailable()
{
    unsigned long result;
    InvokeHelper(0x628, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentVolume()
{
    unsigned long result;
    InvokeHelper(0x629, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetCurrentDiscSide()
{
    unsigned long result;
    InvokeHelper(0x62a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
    NULL);
    return result;
}

BOOL CMediaPlayerDvd::GetCCActive()
{
    BOOL result;
    InvokeHelper(0x62d, DISPATCH_PROPERTYGET, VT_BOOL,
    (void*)&result, NULL);
    return result;
}

void CMediaPlayerDvd::SetCCActive(BOOL bNewValue)
```

```

{
    static BYTE parms[] =
        VTS_BOOL;
    InvokeHelper(0x62d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        bNewValue);
}

unsigned long CMediaPlayerDvd::GetCurrentCCService()
{
    unsigned long result;
    InvokeHelper(0x62c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayerDvd::SetCurrentCCService(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x62c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

CString CMediaPlayerDvd::GetUniqueID()
{
    CString result;
    InvokeHelper(0x630, DISPATCH_PROPERTYGET, VT_BSTR,
        (void*)&result, NULL);
    return result;
}

unsigned long CMediaPlayerDvd::GetColorKey()
{
    unsigned long result;
    InvokeHelper(0x631, DISPATCH_PROPERTYGET, VT_I4, (void*)&result,
        NULL);
    return result;
}

void CMediaPlayerDvd::SetColorKey(unsigned long newValue)
{
    static BYTE parms[] =
        VTS_I4;
    InvokeHelper(0x631, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
        newValue);
}

#include "stdafx.h"
#include "Project Nalay.h"
#include "Main Doc.h"

bool NProcessMessage(long lLineNumber, LPSTR szFilename, HRESULT
hResult, int nMessageType, LPSTR szFunctionName)
{
    CProjectNalayApp * m_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();

```

```

        m_pProjectNalayApp->NProcessMessage(lLineNumber, szFilename,
(HRESULT) hResult, nMessageType, szFunctionName);

        return true;
    }

bool NProcessMessage(long lLineNumber, LPSTR szFilename, WORD
wStringID, int nMessageType, LPSTR szFunctionName)
{
    CProjectNalayApp * m_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    m_pProjectNalayApp->NProcessMessage(lLineNumber, szFilename,
wStringID, nMessageType, szFunctionName);

    return true;
}

void NUpdateAllViews( CView* pSender, LPARAM lHint , CObject* pHint )
{
    CProjectNalayApp * m_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    m_pProjectNalayApp->NUpdateAllViews( pSender, lHint , pHint );
}
// MotionDetectionSettingsDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "MotionDetectionSettingsDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CMotionDetectionSettingsDlg dialog

CMotionDetectionSettingsDlg::CMotionDetectionSettingsDlg(CWnd* pParent
/*=NULL*/)
: CDialog(CMotionDetectionSettingsDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CMotionDetectionSettingsDlg)
    m_Active = FALSE;
    m_DwellTime = 0;
    m_Sensitivity = 0;
    //}}AFX_DATA_INIT
}

void CMotionDetectionSettingsDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);

```

```

        //{AFX_DATA_MAP(CMotionDetectionSettingsDlg)
        DDX_Control(pDX, IDC_SENSITIVITY, m_SensitivityCtrl);
        DDX_Control(pDX, IDC_DWELLTIME, m_DwellTimeCtrl);
        DDX_Check(pDX, IDC_ACTIVE, m_Active);
        DDX_Text(pDX, IDC_DWELLTIME, m_DwellTime);
        DDV_MinMaxUInt(pDX, m_DwellTime, 1, 3600);
        DDX_Slider(pDX, IDC_SENSITIVITY, m_Sensitivity);
        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CMotionDetectionSettingsDlg, CDialog)
    //{AFX_MSG_MAP(CMotionDetectionSettingsDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CMotionDetectionSettingsDlg message handlers

BOOL CMotionDetectionSettingsDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    m_SensitivityCtrl.SetRange(0, 10, TRUE);

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}
// PhoneDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PhoneDlg.h"

#include "TAPIControl.h"
#include "SimpleDirectAudio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CPhoneDlg dialog

CPhoneDlg::CPhoneDlg(CWnd* pParent /*=NULL*/)

```

```

        : CDialog(CPhoneDlg::IDD, pParent)
    {
        //{AFX_DATA_INIT(CPhoneDlg)
        m_AudioFile = _T("");
        m_DialTones = _T("");
        m_PhoneNumber = _T("");
        m_WaitToHangUp = 0;
        //}AFX_DATA_INIT
    }

void CPhoneDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPhoneDlg)
    DDX_Control(pDX, IDC_TEST, m_TestCtrl);
    DDX_Control(pDX, IDC_DIALTONES, m_DialTonesCtrl);
    DDX_Control(pDX, IDC_BROWSE, m_BrowseCtrl);
    DDX_Control(pDX, IDC_AUDIOFILE, m_AudioFileCtrl);
    DDX_Text(pDX, IDC_AUDIOFILE, m_AudioFile);
    DDX_Text(pDX, IDC_DIALTONES, m_DialTones);
    DDX_Text(pDX, IDC_PHONENUMBER, m_PhoneNumber);
    DDX_Text(pDX, IDC_WAITTOHANDUP, m_WaitToHangUp);
    DDV_MinMaxInt(pDX, m_WaitToHangUp, 0, 1000);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPhoneDlg, CDialog)
    //{AFX_MSG_MAP(CPhoneDlg)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    ON_BN_CLICKED(IDC_TEST, OnTest)
    ON_BN_CLICKED(IDC_PLAYAUDIO, OnPlayaudio)
    ON_BN_CLICKED(IDC_RECORDAUDIO, OnRecordaudio)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPhoneDlg message handlers

void CPhoneDlg::OnBrowse()
{
    char BASED_CODE t_szFilter[] = "Audio Files (*.wav)|*.wav|All Files (*.*)|*.*||";
    CString t_strAudioFile;

    t_strAudioFile.Format("%s\\*.wav",
gm_AppConfig.m_DefaultAudioDirectory);

    CFileDialog t_FileDialog(TRUE, ".WAV", t_strAudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);
    // CFileDialog t_FileDialog(TRUE, ".WAV", m_AudioFile,
OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);

    if ( t_FileDialog.DoModal() == IDOK )
    {

```

```

        UpdateData(TRUE);
        m_AudioFile = t_FileDialog.m_ofn.lpstrFile;
        UpdateData(FALSE);
    }
}

void CPhoneDlg::OnTest()
{
    CTAPIControl t_TAPIControl;

    UpdateData();
    t_TAPIControl.QuickCall(gm_AppConfig.m_TAPIDevice, m_PhoneNumber,
m_DialTones, m_AudioFile, m_WaitToHangUp );
}

void CPhoneDlg::OnOK()
{
    UpdateData();
    CDialog::OnOK();
}

BOOL CPhoneDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    CTAPIControl t_TAPIControl;

    if ( !t_TAPIControl.QuickIsTAPIOEnabled(gm_AppConfig.m_TAPIDevice)
)
        m_TestCtrl.EnableWindow(FALSE);

    if (
!t_TAPIControl.QuickIsPlayWAVFileSupported(gm_AppConfig.m_TAPIDevice) )
    {
        m_AudioFileCtrl.EnableWindow(FALSE);
        m_BrowseCtrl.EnableWindow(FALSE);
    }

    if (
!t_TAPIControl.QuickIsGeneratedDTMFSignalsSupported(gm_AppConfig.m_TAPID
evice) )
        m_DialTonesCtrl.EnableWindow(FALSE);

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CPhoneDlg::OnCancel()
{
    if ( AfxMessageBox(IDS_CANCELAREYOUSURE, MB_YESNO |
MB_ICONQUESTION) == IDNO )
        return;
}

```

[illegible]

```

IMPLEMENT_DYNCREATE(CPlaybackDoc, CDocument)

CPlaybackDoc::CPlaybackDoc()
{
}

BOOL CPlaybackDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    return TRUE;
}

CPlaybackDoc::~CPlaybackDoc()
{
}

BEGIN_MESSAGE_MAP(CPlaybackDoc, CDocument)
    //{AFX_MSG_MAP(CPlaybackDoc)
    // NOTE - the ClassWizard will add and remove mapping
    macros here.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPlaybackDoc diagnostics

#ifdef _DEBUG
void CPlaybackDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CPlaybackDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CPlaybackDoc serialization

void CPlaybackDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

```

```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackDoc commands
// PlaybackFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PlaybackFrame.h"
#include "PlaybackVideoView.h"
#include "PlaybackWMPView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CLayout gm_Layout;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackFrame

IMPLEMENT_DYNCREATE(CPlaybackFrame, CMDIChildWnd)

CPlaybackFrame::CPlaybackFrame()
{
}

CPlaybackFrame::~CPlaybackFrame()
{
}

BEGIN_MESSAGE_MAP(CPlaybackFrame, CMDIChildWnd)
    //{AFX_MSG_MAP(CPlaybackFrame)
    ON_WM_CREATE()
    ON_WM_CLOSE()
    ON_WM_DESTROY()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackFrame message handlers

BOOL CPlaybackFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // create a splitter with 2 rows, 1 column
    if (!m_wndSplitter.CreateStatic(this, 1, 2))
    {
        TRACE0("Failed to CreateStaticSplitter\n");
    }

```

```

        return FALSE;
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
        pContext->m_pNewViewClass, CSize(350, 50), pContext))
    {
        TRACE0("Failed to create first pane\n");
        return FALSE;
    }

    // add the second splitter pane - an input view in row 0
    if (!m_wndSplitter.CreateView(0, 1,
        RUNTIME_CLASS(CPlaybackWMPView), CSize(0, 0), pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(0,0));

    return TRUE;
}

BOOL CPlaybackFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.x = gm_Layout.m_rectPlaybackWindow.left;
    cs.y = gm_Layout.m_rectPlaybackWindow.top;
    cs.cx = gm_Layout.m_rectPlaybackWindow.Width();
    cs.cy = gm_Layout.m_rectPlaybackWindow.Height();

    return CMDIChildWnd::PreCreateWindow(cs);
}

int CPlaybackFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_VIDEOPLAYBACK))
    {
        return FALSE;        // fail to create
    }

    return 0;
}

void CPlaybackFrame::OnClose()
{
    gm_Layout.m_bPlaybackWindowOpen = FALSE;

    CMDIChildWnd::OnClose();
}

```

```

void CPlaybackFrame::OnDestroy()
{
    CMDIChildWnd::OnDestroy();

    GetWindowRect(&gm_Layout.m_rectPlaybackWindow);
    GetParent()->ScreenToClient(&gm_Layout.m_rectPlaybackWindow);
}
// PlaybackListView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PlaybackListView.h"
#include "PlaybackDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CPlaybackListView

IMPLEMENT_DYNCREATE(CPlaybackListView, CListView)

CPlaybackListView::CPlaybackListView()
{
}

CPlaybackListView::~CPlaybackListView()
{
}

BEGIN_MESSAGE_MAP(CPlaybackListView, CListView)
    //{AFX_MSG_MAP(CPlaybackListView)
    ON_COMMAND(ID_PLAYBACK_FASTFORWARD, OnPlaybackFastforward)
    ON_COMMAND(ID_PLAYBACK_FORWARD, OnPlaybackForward)
    ON_COMMAND(ID_PLAYBACK_GO, OnPlaybackGo)
    ON_COMMAND(ID_PLAYBACK_PAUSE, OnPlaybackPause)
    ON_COMMAND(ID_PLAYBACK_STOP, OnPlaybackStop)
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
    ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLargeicons)
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
    ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallicons)
    ON_COMMAND(ID_OPEN_VIDEO_FILE, OnOpenVideoFile)
    ON_NOTIFY_REFLECT(NM_DBLCLK, OnDblclk)
    ON_WM_CONTEXTMENU()
    ON_COMMAND(ID_PLAY_SELECTED_VIDEO, OnPlaySelectedVideo)
    ON_COMMAND(ID_DELETE_VIDEO_FILE, OnDeleteVideoFile)
    ON_COMMAND(ID_UPDATE_VIDEO_PLAYBACK, OnUpdateVideoPlayback)
    ON_COMMAND(ID_OPEN_VIDEO_DIRECTORY, OnOpenVideoDirectory)
    //}}AFX_MSG_MAP

```

```

END_MESSAGE_MAP()

////////////////////////////////////
////
// CPlaybackListView drawing

void CPlaybackListView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
////
// CPlaybackListView diagnostics

#ifdef _DEBUG
void CPlaybackListView::AssertValid() const
{
    CListView::AssertValid();
}

void CPlaybackListView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
////
// CPlaybackListView message handlers

BOOL CPlaybackListView::PreCreateWindow(CREATESTRUCT& cs)
{
    // Default to report view
    cs.style |= LVS_LIST | LVS_NOSORTHEADER | LVS_SINGLESEL;

    return CListView::PreCreateWindow(cs);
}

void CPlaybackListView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    m_LargeImageList.Create(IDB_VIDEOPLAYBACKLARGEICONS, 32, 1,
    RGB(255, 255, 255));
    m_SmallImageList.Create(IDB_VIDEOPLAYBACKSMALLICONS, 16, 1,
    RGB(255, 255, 255));
    // m_StateImageList.Create(IDB_CONNECTIONSSTATEICONS, 8, 1, RGB(255,
    0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    // t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

```

```

//      t_strItem.LoadString(IDS_LABEL);
//      t_strItem.LoadString(IDS_CONNECTION);
//      t_ctlList.InsertColumn(0, t_strItem);
//      SetColumnWidth(0, 80);
//      t_strItem.LoadString(IDS_DATE);
//      t_ctlList.InsertColumn(1, t_strItem);
//      SetColumnWidth(1, 110);
//      t_strItem.LoadString(IDS_STARTTIME);
//      t_ctlList.InsertColumn(2, t_strItem);
//      SetColumnWidth(2, 80);
//      t_strItem.LoadString(IDS_STOPTIME);
//      t_ctlList.InsertColumn(3, t_strItem);
//      SetColumnWidth(3, 80);
//      t_strItem.LoadString(IDS_FILENAME);
//      t_ctlList.InsertColumn(4, t_strItem);
//      SetColumnWidth(4, 110);

// Start out in details view
SetViewType(LVS_REPORT);

// Insert item
LoadVideoPlaybackFiles();
}

void CPlaybackListView::OnPlaybackFastforward()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL,
NUPDATE_PLAYBACKFASTFORWARD, NULL);
}

void CPlaybackListView::OnPlaybackForward()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL,
NUPDATE_PLAYBACKFORWARD, NULL);
}

void CPlaybackListView::OnPlaybackGo()
{
    OnPlaySelectedVideo();
/*
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL, NUPDATE_PLAYBACKGO,
NULL);
*/
}

void CPlaybackListView::OnPlaybackPause()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )

```

```

        t_pPlaybackDoc->UpdateAllViews(NULL, NUPDATE_PLAYBACKPAUSE,
NULL);
    }

void CPlaybackListView::OnPlaybackStop()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL, NUPDATE_PLAYBACKSTOP,
NULL);
}

void CPlaybackListView::OnOpenVideoFile()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL, NUPDATE_OPENVIDEOFILE,
NULL);
}

void CPlaybackListView::OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint)
{
    switch ( lHint )
    {
    case NUPDATE_VIEWDETAILS:
        OnViewDetails();
        break;
    case NUPDATE_VIEWLARGEICONS:
        OnViewLargeicons() ;
        break;
    case NUPDATE_VIEWLIST:
        OnViewList() ;
        break;
    case NUPDATE_VIEWSMALLICONS:
        OnViewSmallicons();
        break;
    case NUPDATE_DELETEVIDEOFILE:
        OnDeleteVideoFile();
        break;
    }
}

void CPlaybackListView::OnViewSmallicons()
{
    if (GetViewType() != LVS_SMALLICON)
        SetViewType(LVS_SMALLICON);
}

void CPlaybackListView::OnViewDetails()
{
    if (GetViewType() != LVS_REPORT)
        SetViewType(LVS_REPORT);
}

void CPlaybackListView::OnViewLargeicons()
{

```

```

        if (GetViewType() != LVS_ICON)
            SetViewType(LVS_ICON);
    }

void CPlaybackListView::OnViewList()
{
    if (GetViewType() != LVS_LIST)
        SetViewType(LVS_LIST);
}

BOOL CPlaybackListView::SetViewType(DWORD dwViewType)
{
    return(ModifyStyle(LVS_TYPEMASK, dwViewType & LVS_TYPEMASK));
}

DWORD CPlaybackListView::GetViewType()
{
    return(GetStyle() & LVS_TYPEMASK);
}

BOOL CPlaybackListView::SetColumnWidth(int Column, int Width)
{
    CListCtrl& t_ctlList = GetListCtrl();
    LV_COLUMN t_lvColumn;

    t_lvColumn.mask = LVCF_WIDTH;
    t_lvColumn.cx = Width;
    return t_ctlList.SetColumn(Column, &t_lvColumn);
}

bool CPlaybackListView::LoadVideoPlaybackFiles()
{
    int i;
    CString t_str;
    CListCtrl& t_ctlList = GetListCtrl();
    CStringArray t_arrFiles;

    // Clear contents of control
    t_ctlList.DeleteAllItems();

    // Get list of AVI files
    GetVideoPlaybackFiles(t_arrFiles);

    // Cycle through files and populate list
    for ( i = 0; i < t_arrFiles.GetSize(); i++ )
    {
        CString t_str;
        char t_szPath[_MAX_PATH];
        char t_szPath2[_MAX_PATH];
        LPSTR t_szFilename;
        char * t_pNextToken;
        int t_nRecordTrigger = 0;
        CString t_strDate;
        CString t_strStartTime;
        CString t_strDuration;

        // Copy path to token variable

```

```

lstrcpy(t_szPath, (LPCTSTR) t_arrFiles.ElementAt(i));
PathRemoveExtension(t_szPath);

// Get and insert Label
// First strip out the path
t_pNextToken = strtok(t_szPath, "~");
if ( t_pNextToken == NULL )
    continue;

// Get just the filename as the label, which corresponds to
the camera label
lstrcpy( t_szPath2, t_pNextToken );
t_szFilename = PathFindFileName(t_szPath2);

// Get and insert Date
t_pNextToken = strtok(NULL, "~");
if ( t_pNextToken != NULL )
{
    t_strDate = t_pNextToken ;

    // Get and insert Start Time
    t_pNextToken = strtok(NULL, "~");
    if ( t_pNextToken != NULL )
    {
        t_strStartTime = t_pNextToken ;
        t_strStartTime.Replace('$', ':');

        // Get and insert Duration
        t_pNextToken = strtok(NULL, "~");
        if ( t_pNextToken != NULL )
        {
            t_strDuration = t_pNextToken ;
            t_strDuration.Replace('$', ':');

            // Get RecordTrigger value to set icon
            t_pNextToken = strtok(NULL, "~");
            if ( t_pNextToken != NULL )
                t_nRecordTrigger =
atoi(t_pNextToken);
        }
    }
}

// Update list
t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
                    i, (LPCTSTR) t_szFilename ,
                    NULL, NULL,
t_nRecordTrigger,
                    NULL);
t_ctlList.SetItemText(i, 1, t_strDate );
t_ctlList.SetItemText(i, 2, t_strStartTime);
t_ctlList.SetItemText(i, 3, t_strDuration);

// Insert full filename
t_ctlList.SetItemText(i, 4, (LPCTSTR)
t_arrFiles.ElementAt(i) );

```

```

    }

    return true;
}

bool CPlaybackListView::GetVideoPlaybackFiles(CStringArray& arrFiles)
{
    CFileFind t_FileFind;
    BOOL t_bWorking;
    char t_szSearchPath[_MAX_PATH];

    // Build path
    lstrcpy(t_szSearchPath,
gm_AppConfig.m_DefaultVideoPlaybackDirectory);
    PathAppend(t_szSearchPath, "*.wmv");

    // Start search
    t_bWorking = t_FileFind.FindFile(t_szSearchPath);
    while (t_bWorking)
    {
        t_bWorking = t_FileFind.FindNextFile();
        arrFiles.Add(t_FileFind.GetFilePath());
    }

    return true;
}

void CPlaybackListView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    OnPlaySelectedVideo();

    *pResult = 0;
}

void CPlaybackListView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    DisplayContextMenu(this, point, IDR_POPUP_VIDEOPLAYBACK);
}

int CPlaybackListView::GetSelectedItem()
{
    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem = -1;

    // Get selected item - this is a single-selection list control
    POSITION t_pos = t_ctlList.GetFirstSelectedItemPosition();

    // Validate that an item was selected
    if (t_pos == NULL)
    {
        NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_WARNING,
"PlaybackListView", IDS_NOVIDEOSELECTED);
        goto Exit1;
    }

    // Get the item number selected
    t_nItem = t_ctlList.GetNextSelectedItem(t_pos);
}

```

```

Exit1:
    return t_nItem;
}

bool CPlaybackListView::GetSelectedFilename(CString& strFilename)
{
    bool t_bReturn = false;

    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem ;

    // Get selected item - this is a single-selection list control
    t_nItem = GetSelectedItem();
    if ( t_nItem == -1 )
        goto Exit1;

    // Get the m_Connections array index for that item
    strFilename = t_ctlList.GetItemText(t_nItem, 4);

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

void CPlaybackListView::OnPlaySelectedVideo()
{
    CString t_strFilename;
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();

    if ( GetSelectedFilename(t_strFilename) )
    {
        if ( t_pPlaybackDoc != NULL )
        {
            t_pPlaybackDoc->m_strVideoFile = t_strFilename;
            t_pPlaybackDoc->UpdateAllViews(NULL,
NUDATE_PLAYSELECTEDVIDEOFILE, NULL);
        }
    }
}

void CPlaybackListView::OnDeleteVideoFile()
{
    CString t_strFilename;

    if ( GetSelectedFilename(t_strFilename) )
    {
        DeleteFile(t_strFilename);
        LoadVideoPlaybackFiles();
    }
}

void CPlaybackListView::OnUpdateVideoPlayback()
{
    LoadVideoPlaybackFiles();
}

```

```

}

void CPlaybackListView::OnOpenVideoDirectory()
{
    char t_szPath[_MAX_PATH];

    lstrcpy(t_szPath, gm_AppConfig.m_DefaultVideoPlaybackDirectory);
    if ( PromptForFolder(m_hWnd, "Select Default Video Playback
Path", t_szPath) )
    {
        gm_AppConfig.m_DefaultVideoPlaybackDirectory = t_szPath;
        LoadVideoPlaybackFiles();
    }
}

// PlaybackVideoView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PlaybackVideoView.h"
#include "PlaybackDoc.h"

#include "direct.h" // For directory functions

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CAppConfig gm_AppConfig;

////////////////////////////////////
/////
// CPlaybackVideoView

IMPLEMENT_DYNCREATE(CPlaybackVideoView, CView)

CPlaybackVideoView::CPlaybackVideoView()
{
}

CPlaybackVideoView::~CPlaybackVideoView()
{
}

BEGIN_MESSAGE_MAP(CPlaybackVideoView, CView)
    ///{(AFX_MSG_MAP(CPlaybackVideoView)
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_SIZE()
    ON_COMMAND(ID_OPEN_VIDEO_FILE, OnOpenVideoFile)
    ON_COMMAND(ID_PLAYBACK_FORWARD, OnPlaybackForward)
    ON_COMMAND(ID_PLAYBACK_GO, OnPlaybackGo)
    ON_COMMAND(ID_PLAYBACK_PAUSE, OnPlaybackPause)
    ON_COMMAND(ID_PLAYBACK_STOP, OnPlaybackStop)

```

```

ON_COMMAND(ID_PLAYBACK_FASTFORWARD, OnPlaybackFastforward)
ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLarg icons)
ON_COMMAND(ID_VIEW_LIST, OnViewList)
ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallicons)
ON_WM_MOVE()
ON_WM_CONTEXTMENU()
ON_COMMAND(ID_PLAY_SELECTED_VIDEO, OnPlaySelectedVideo)
ON_COMMAND(ID_DELETE_VIDEO_FILE, OnDeleteVideoFile)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPlaybackVideoView drawing

void CPlaybackVideoView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

////////////////////////////////////
/////
// CPlaybackVideoView diagnostics

#ifdef _DEBUG
void CPlaybackVideoView::AssertValid() const
{
    CView::AssertValid();
}

void CPlaybackVideoView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
/////
// CPlaybackVideoView message handlers

void CPlaybackVideoView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
    case NUUPDATE_PLAYSELECTEDVIDEOFILE:
        PlaySelectedVideoFile();
        break;
    case NUUPDATE_OPENVIDEOFILE:
        OnOpenVideoFile();
        break;
    case NUUPDATE_PLAYBACKFORWARD:
        OnPlaybackForward();
        break;
    case NUUPDATE_PLAYBACKGO:

```

```

        OnPlaybackGo();
        break;
    case NUUPDATE_PLAYBACKPAUSE:
        OnPlaybackPause();
        break;
    case NUUPDATE_PLAYBACKSTOP:
        OnPlaybackStop();
        break;
    case NUUPDATE_PLAYBACKFASTFORWARD:
        OnPlaybackFastforward();
        break;
    }
}

int CPlaybackVideoView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    m_SimpleVideo.InitializeVideoPlayback();

    return 0;
}

void CPlaybackVideoView::OnDestroy()
{
    CView::OnDestroy();

    m_SimpleVideo.UninitializeVideoPlayback();
}

void CPlaybackVideoView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    CRect rect;

    GetClientRect(&rect);
    m_SimpleVideo.VideoWindowSetWindowPos(rect);
}

void CPlaybackVideoView::OnOpenVideoFile()
{
    char BASED_CODE t_szFilter[] = "Video Files (*.avi)|*.avi|All Files (*.*)|*.*||";
    CString t_strVideoFile;

    CFileDialog t_FileDialog(TRUE, ".AVI", t_strVideoFile,
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, t_szFilter);

    // Adjust directory to point to Video Playback directory

    // Open a new file
    if ( t_FileDialog.DoModal() == IDOK )
    {

```

```

        CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* )
GetDocument();

        if ( t_pPlaybackDoc != NULL )
        {
            t_pPlaybackDoc->m_strVideoFile =
t_FileDialog.m_ofn.lpstrFile;

            // If new file is selected, play new file
            m_SimpleVideo.VideoPlaybackStart(t_pPlaybackDoc-
>m_strVideoFile , m_hWnd);
        }
    }

bool CPlaybackVideoView::PlaySelectedVideoFile()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();

    if ( t_pPlaybackDoc != NULL )
        m_SimpleVideo.VideoPlaybackStart(t_pPlaybackDoc-
>m_strVideoFile , m_hWnd);

    return true;
}

void CPlaybackVideoView::OnPlaybackForward()
{
    m_SimpleVideo.VideoPlaybackSetRate((double)
gm_AppConfig.m_ForwardSpeed);
}

void CPlaybackVideoView::OnPlaybackGo()
{
    m_SimpleVideo.VideoPlaybackSetRate((double) 1);
    m_SimpleVideo.VideoPlaybackStart(m_hWnd);
}

void CPlaybackVideoView::OnPlaybackPause()
{
    m_SimpleVideo.VideoPlaybackTogglePause();
    m_SimpleVideo.VideoPlaybackSetRate((double) 1);
}

void CPlaybackVideoView::OnPlaybackStop()
{
    m_SimpleVideo.VideoPlaybackStop();
}

void CPlaybackVideoView::OnPlaybackFastforward()
{
    m_SimpleVideo.VideoPlaybackSetRate((double)
gm_AppConfig.m_FastForwardSpeed);
}

void CPlaybackVideoView::OnViewDetails()

```

```

{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL, NUUPDATE_VIEWDETAILS,
NULL);
}

void CPlaybackVideoView::OnViewLargeicons()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL,
NUUPDATE_VIEWLARGEICONS, NULL);
}

void CPlaybackVideoView::OnViewList()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL, NUUPDATE_VIEWLIST,
NULL);
}

void CPlaybackVideoView::OnViewSmallicons()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL,
NUUPDATE_VIEWSMALLICONS, NULL);
}

void CPlaybackVideoView::OnMove(int x, int y)
{
    CView::OnMove(x, y);
    CRect rect;

    GetClientRect(&rect);
    m_SimpleVideo.VideoWindowSetWindowPos(rect);
}

LRESULT CPlaybackVideoView::DefWindowProc(UINT message, WPARAM wParam,
LPARAM lParam)
{
    m_SimpleVideo.VideoWindowNotifyOwnerMessage(m_hWnd, message,
wParam, lParam);
    return CView::DefWindowProc(message, wParam, lParam);
}

void CPlaybackVideoView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    DisplayContextMenu(this, point, IDR_POPUP_VIDEOPLAYBACK);
}

void CPlaybackVideoView::OnPlaySelectedVideo()
{
    PlaySelectedVideoFile();
}

```

```

}

void CPlaybackVideoView::OnDeleteVideoFile()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();
    if ( t_pPlaybackDoc != NULL )
        t_pPlaybackDoc->UpdateAllViews(NULL,
NUPDATE_DELETEVIDEOFILE, NULL);
}
// PlaybackWMPView.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PlaybackWMPView.h"
#include "PlaybackDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPlaybackWMPView

IMPLEMENT_DYNCREATE(CPlaybackWMPView, CFormView)

CPlaybackWMPView::CPlaybackWMPView()
: CFormView(CPlaybackWMPView::IDD)
{
    ///{{AFX_DATA_INIT(CPlaybackWMPView)
    ///}}AFX_DATA_INIT
    m_bIsWMPInitialized = false;
}

CPlaybackWMPView::~CPlaybackWMPView()
{
}

void CPlaybackWMPView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CPlaybackWMPView)
    DDX_Control(pDX, IDC_MEDIAPLAYER, m_WMP);
    ///}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPlaybackWMPView, CFormView)
    ///{{AFX_MSG_MAP(CPlaybackWMPView)
    ON_WM_MOVE()
    ON_WM_SIZE()
    ON_COMMAND(ID_OPEN_VIDEO_FILE, OnOpenVideoFile)
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackWMPView diagnostics

#ifdef _DEBUG
void CPlaybackWMPView::AssertValid() const
{
    CFormView::AssertValid();
}

void CPlaybackWMPView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPlaybackWMPView message handlers

void CPlaybackWMPView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();

    m_bIsWMPInitialized = true;
    m_WMP.SetShowStatusBar(TRUE);
    m_WMP.SetShowPositionControls(FALSE);
    m_WMP.SetEnableTracker(TRUE);
    ResizeWMP();
}

void CPlaybackWMPView::OnMove(int x, int y)
{
    CFormView::OnMove(x, y);

    ResizeWMP();
}

void CPlaybackWMPView::OnSize(UINT nType, int cx, int cy)
{
    CFormView::OnSize(nType, cx, cy);

    ResizeWMP();
}

void CPlaybackWMPView::ResizeWMP()
{
    if ( m_bIsWMPInitialized )
    {
        CRect rect;
        GetClientRect(&rect);
        m_WMP.MoveWindow(&rect);
    }
}

```

```

void CPlaybackWMPView::OnOpenVideoFile()
{
    char BASED_CODE t_szFilter[] = "Windows Media Files
(*.wmv)|*.wmv|All Files (*.*)|*.*||";

    CFileDialog t_FileDialog(TRUE, ".WMV", "", OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT, t_szFilter);

    if ( t_FileDialog.DoModal() == IDOK )
    {
        m_WMP.SetFileName(t_FileDialog.m_ofn.lpstrFile );
        ResizeWMP();
    }
}

void CPlaybackWMPView::OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint)
{
    switch ( lHint )
    {
        case NUPDATE_PLAYSELECTEDVIDEOFILE:
            PlaySelectedVideoFile();
            break;
        case NUPDATE_OPENVIDEOFILE:
            OnOpenVideoFile();
            break;
    }
}

bool CPlaybackWMPView::PlaySelectedVideoFile()
{
    CPlaybackDoc* t_pPlaybackDoc = (CPlaybackDoc* ) GetDocument();

    if ( t_pPlaybackDoc != NULL )
    {
        m_WMP.SetFileName(t_pPlaybackDoc->m_strVideoFile );
    }

    return true;
}
// Project Nalay.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Project Nalay.h"

#include "errors.h"
#include "SimpleVideo.h"
#include "DirectPlay.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "Main Doc.h"
#include "Main View.h"
#include "ConnectionsListView.h"
#include "ConnectionsFrame.h"
#include "FeedbackFrame.h"

```

```

#include "FeedbackListView.h"
#include "FeedbackStatusView.h"
#include "InstantMessengerDoc.h"
#include "InstantMessengerFrame.h"
#include "InstantMessengerConversationView.h"
#include "LocalVideoFrame.h"
#include "LocalVideoEventsView.h"
#include "PlaybackDoc.h"
#include "PlaybackFrame.h"
#include "PlaybackListView.h"
#include "LocalVideoDoc.h"
#include "PropPageEMail.h"
#include "PropPageConnections.h"
#include "PropPageFeedback.h"
#include "PropPageYellowPages.h"
#include "PropPageVideo.h"
#include "PropPageAudio.h"
#include "PropPageSecurityPrompts.h"
#include "SimpleMAPI.h"
#include "PropPagePhoneModem.h"
#include "PropPageRecordVideo.h"
#include "PropPageVideoPlayback.h"
#include "PropPageUpdate.h"
#include "RemoteVideoFrame.h"
#include "RemoteVideoEventsView.h"
#include "AppSecurityDlg.h"

#include "direct.h"
#include "winsock2.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_SERIAL( CLayout, CObject, 2 )
IMPLEMENT_SERIAL( CLayoutConnectionWindow, CObject, 2 )
IMPLEMENT_SERIAL( CAppConfig, CObject, 2 )

//-----
// Name: gm_Connections
// Desc: Global access array of connections
//
//-----
CConnectionsArray gm_Connections;

//-----
// Name: gm_csHostList
// Desc: Global critical section holder
//
//-----
CRITICAL_SECTION gm_csHostList;

```

```

CRITICAL_SECTION gm_csEmailAction;
CRITICAL_SECTION gm_csAudioAction;
CRITICAL_SECTION gm_csPhoneAction;
CRITICAL_SECTION gm_csX10Action;
CRITICAL_SECTION gm_csRecordVideo;

//-----
// Name: gm_Layout
// Desc: Global layout info
//
//-----
CLayout gm_Layout;

//-----
// Name: gm_AppConfig
// Desc: Global configuration info
//
//-----
CAppConfig gm_AppConfig;

// Data type and method for getting version info
typedef struct
{
    CString strCompanyName;
    CString strFileDescription;
    CString strFileVersion;
    CString strInternalName;
    CString strLegalCopyright;
    CString strOriginalFilename;
    CString strProductName;
    CString strProductVersion;
    CString strComments;
    CString strLegalTrademarks;
    CString strPrivateBuild;
    CString strSpecialBuild;
} NVersionInfo;

//-----
// Name: SendFeedbackMessage
// Desc: Global feedback message handling routine
//
//-----
bool GetFeedbackFileName(CString& strFileName);
void SendFeedbackMessage(long lLineNumber, LPSTR szFilename, int nType,
    CString strMessage, CString strDescription)
{
    CProjectNalayApp * t_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    if ( t_pProjectNalayApp->m_pFeedbackListView != NULL )

```

```

        t_pProjectNalayApp->m_pFeedbackListView-
>AddMessage(lLineNumber, szFilename, nType, strMessage,
strDescription);

        // Write feedback message to feedback file
        CStdioFile t_File;
        CString t_strDefaultFilename;
        CString t_strBuffer;
        TCHAR t_szJustFileName[_MAX_PATH];
        TCHAR t_szFileName[_MAX_PATH];
        CTime t_curTime;
        CString t_strTime;

        // Get current time
        t_curTime = CTime::GetCurrentTime();
        t_strTime = t_curTime.Format("%c");

        // Get just the filename
        lstrcpy(t_szFileName, szFilename);
        lstrcpy(t_szJustFileName, PathFindFileName(t_szFileName));
        PathRemoveExtension(t_szJustFileName);

        GetFeedbackFileName(t_strDefaultFilename);
        if ( !t_File.Open(t_strDefaultFilename, CFile::modeWrite |
CFile::typeText | CFile::modeCreate | CFile::modeNoTruncate ) )
            return ;

        t_File.SeekToEnd();
        t_strBuffer.Format("%d,%s,%s,%s,%s,%ld\n", nType, strMessage,
strDescription, t_strTime, t_szJustFileName, lLineNumber);
        t_File.WriteString(t_strBuffer);
        t_File.Flush();
        t_File.Close();
    }

void SendFeedbackHResult(long lLineNumber, LPSTR szFilename, int nType,
CString strMessage, HRESULT hResult)
{
    TCHAR buffer[128];

    // Retrieve the text associated with the HRESULT value, or
    // if there is no error text ensure that the buffer is blank
    if ( AMGetErrorText(hResult, buffer, 127 ) == 0 )
        ltoa((long) hResult, buffer, 10);

    SendFeedbackMessage(lLineNumber, szFilename, nType, strMessage,
buffer);
}

void SendFeedbackIDS(long lLineNumber, LPSTR szFilename, int nType,
CString strMessage, UINT nStringID)
{
    CString t_strDescription;
    t_strDescription.LoadString(nStringID);

```

```

        SendFeedbackMessag (lLineNumber, szFilename, nType, strMessage,
t_strDescription);
    }

void NUpdateAllViews( CView* pSender, LPARAM lHint , CObject* pHint )
{
    CProjectNalayApp * t_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    t_pProjectNalayApp->NUpdateAllViews( pSender, lHint , pHint );
}

void SendIMMessage(CConnectionMsg* pConnectionMsg)
{
    CProjectNalayApp * t_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    t_pProjectNalayApp->NUpdateAllViews( NULL,
NUPDATE_IMMESSAGERECEIVED, (CObject*) pConnectionMsg);
}

void SendVSMMessage(DWORD dwMessageId, CConnectionMsg* pConnectionMsg )
{
    CProjectNalayApp * t_pProjectNalayApp = (CProjectNalayApp*)
AfxGetApp();
    t_pProjectNalayApp->NUpdateAllViews( NULL, dwMessageId,
(CObject*) pConnectionMsg);
}

bool GetFeedbackFileName(CString& strFileName)
{
    CString t_strDefaultFilename;
    char t_szSavePath[_MAX_PATH];

    // Create save path
    t_strDefaultFilename.LoadString(IDS_DEFAULTFEEDBACKFILENAME);
    lstrcpy(t_szSavePath, gm_AppConfig.m_StartupPath);
    PathAppend(t_szSavePath, t_strDefaultFilename);

    strFileName = t_szSavePath;
    return true;
}

//
//-----
//-----
//-----
//
// Function: PromptForFolder
// Purpose: Prompts user to select a folder
// Return: true = success, in which case selectedFolderPath set to
folder
//         selected by user
//

```

```

//-----
//-----
#define BOOL2bool(value) (value ? true : false)
#define bool2BOOL(value) (value == true ? TRUE : FALSE)
bool PromptForFolder(HWND hWnd, char *promptText, char
*selectedFolderPath)
{
    HRESULT          hr;
    LPITEMIDLIST pidl;
    BROWSEINFO  bi;
    char        selectedFolder[_MAX_PATH];
    BOOL        bPathOK;
    LPALLOC     pMalloc;

    bi.hwndOwner = hWnd;
    bi.pidlRoot = NULL;
    bi.pszDisplayName = selectedFolder;
    bi.lpszTitle = promptText;
    bi.ulFlags = BIF_RETURNONLYFSDIRS; // | BIF_USENEWUI
    bi.lpfn = NULL;
    bi.lParam = NULL;
    bi.iImage = 0;

    pidl = SHBrowseForFolder(&bi);
    if (! pidl)
    {
        return false;
    }

    bPathOK = SHGetPathFromIDList(pidl, selectedFolderPath);

    hr = SHGetMalloc(&pMalloc);
    if SUCCEEDED(hr)
    {
        pMalloc->Free(pidl);
        pMalloc->Release();
    }

    return BOOL2bool(bPathOK);
}
//
//-----
//-----
// Function: DisplayContextMenu
// Purpose: Prompts user with a context sensitive menu
// Return: true = success
//
//-----
void DisplayContextMenu(CWnd* pWnd, CPoint point, UINT IDResource)
{
    CMenu t_mnuMain;
    CPoint t_point = point;

```

```

    if ( t_mnuMain.LoadMenu(IDResource) )
    {
        CMenu * t_pmnuPopup;
        t_pmnuPopup = t_mnuMain.GetSubMenu(0);

        if ( t_pmnuPopup != NULL )
            t_pmnuPopup->TrackPopupMenu(TPM_LEFTALIGN |
TPM_RIGHTBUTTON, t_point.x, t_point.y, pWnd);
    }
}
//
//-----

//-----
//
// Function: GetMostRecentVideoFile
// Purpose: Gets the latest video from the Playback directory
// Return: true = success
//
//-----
bool GetMostRecentVideoFile(CString& t_strVideoFile)
{
    CFileFind t_FileFind;
    BOOL t_bWorking;
    char t_szSearchPath[_MAX_PATH];
    CFileStatus t_FileStatusOld;
    CFileStatus t_FileStatusNew;
    CString t_strTempVideoFile;

    t_strTempVideoFile.LoadString(IDS_TEMPWMVRECORDFILENAME);
    t_FileStatusOld.m_mtime = CTime();

    // Build path
    lstrcpy(t_szSearchPath,
gm_AppConfig.m_DefaultVideoPlaybackDirectory);
    PathAppend(t_szSearchPath, "*.wmv");

    // Start search
    t_bWorking = t_FileFind.FindFile(t_szSearchPath);
    while (t_bWorking)
    {
        t_bWorking = t_FileFind.FindNextFile();

        // Make sure this is not the temp file
        if ( t_strTempVideoFile == t_FileFind.GetFileName() )
            continue;

        CFile::GetStatus(t_FileFind.GetFilePath(),
t_FileStatusNew);
        if ( t_FileStatusNew.m_mtime > t_FileStatusOld.m_mtime )
        {
            t_strVideoFile = t_FileFind.GetFilePath();

```

```

        t_FileStatusOld = t_FileStatusNew;
    }
}

return true;
}
//
//-----

//-----
//
// Function: GetCurrentIPAddress
// Purpose: Gets the IP address of the local machine
// Return: CString - IP Address 111.111.111.111
//
//-----
CString GetCurrentIPAddress()
{
    CString    strIPAddress;
    WSADATA    wsad;
    int        err;
    char        szHostName[1024];
    HOSTENT*   pHE;
    IN_ADDR    inaddr;

    err = WSASStartup(MAKEWORD(1,1), &wsad);
    if (err) return CString();

    err = gethostname(szHostName, 1024);
    if (err) goto end;

    pHE = gethostbyname(szHostName);
    if (!pHE) goto end;

    memcpy(&inaddr, pHE->h_addr, 4);
    strIPAddress = inet_ntoa(inaddr);

end:
    WSACleanup();
    return strIPAddress;
}
//
//-----

//-----
//
// Function: PromptSecurityPassword
// Purpose: Prompts user if necessary for app password

```

```

// Return: true if no password required or user entered correct
password
//
//-----
bool PromptSecurityPassword(UINT AppSecurityLoc)
{
    bool t_bReturn = false;
    CAppSecurityDlg t_AppSecurityDlg;
    int i;

    // If there is no requirement for a password, simply continue
    if ( gm_AppConfig.m_EnableAppSecurityPrompt == FALSE )
    {
        if ( AppSecurityLoc == PWD_PROMPT_ON_EXIT )
        {
            CString t_strMessage;

            if ( gm_AppConfig.m_PromptOnExit )
            {
                CString t_strTitle;
                t_strTitle.LoadString(IDR_MAINFRAME);
                t_strMessage.LoadString(IDS_EXIT_PROMPT);
                if ( MessageBox(GetFocus(), t_strMessage,
t_strTitle, MB_OKCANCEL) == IDOK )
                    t_bReturn = true;
            }
            else
                t_bReturn = true;
        }
        else
            t_bReturn = true;

        goto Exit1;
    }

    // Check if app security is required at the requested location
    switch ( AppSecurityLoc )
    {
    case PWD_PROMPT_ON_LAUNCH:
        if ( gm_AppConfig.m_PwdPromptOnLaunch == FALSE )
        {
            t_bReturn = true;
            goto Exit1;
        }
        break;
    case PWD_PROMPT_ON_EXIT:
        if ( gm_AppConfig.m_PwdPromptOnExit == FALSE )
        {
            t_bReturn = true;
            goto Exit1;
        }
        break;
    case PWD_PROMPT_ON_CONNECTION:
        if ( gm_AppConfig.m_PwdPromptOnConnection == FALSE )
        {
            t_bReturn = true;
        }
    }
}

```

```

        goto Exit1;
    }
    break;
case PWDPROMPT_ONALARM:
    if ( gm_AppConfig.m_PwdPromptOnAlarm == FALSE )
    {
        t_bReturn = true;
        goto Exit1;
    }
    break;
case PWDPROMPT_ONCONFIGURATION:
    if ( gm_AppConfig.m_PwdPromptOnConfiguration == FALSE )
    {
        t_bReturn = true;
        goto Exit1;
    }
    break;
};

// If the password is empty then don't bother checking
if ( gm_AppConfig.m_AppPassword.IsEmpty() )
{
    t_bReturn = true;
    goto Exit1;
}

// Give users three chances
for ( i = 0; i < 3; i++ )
{
    if ( t_AppSecurityDlg.DoModal() == IDOK )
    {
        CString t_strMessage;

        // Only set return value to true if the user matches
password
        if ( gm_AppConfig.m_AppPassword ==
t_AppSecurityDlg.m_Password )
        {
            t_bReturn = true;
            goto Exit1;
        }

        t_strMessage.LoadString(IDS_INCORRECTPASSWORD);
        MessageBox(GetFocus(), t_strMessage, NULL, MB_OK |
MB_ICONWARNING );
    }
    else
        goto Exit1;
}

Exit1:
    return t_bReturn;
}

////////////////////////////////////
////////

```

```

// CProjectNalayApp

BEGIN_MESSAGE_MAP(CProjectNalayApp, CWinApp)
//{{AFX_MSG_MAP(CProjectNalayApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_VIEW_CONNECTIONS, OnViewConnections)
    ON_UPDATE_COMMAND_UI(ID_VIEW_CONNECTIONS,
OnUpdateViewConnections)
    ON_COMMAND(ID_VIEW_FEEDBACK, OnViewFeedback)
    ON_UPDATE_COMMAND_UI(ID_VIEW_FEEDBACK, OnUpdateViewFeedback)
    ON_COMMAND(ID_VIEW_VIDEO_PLAYBACK, OnViewVideoPlayback)
    ON_UPDATE_COMMAND_UI(ID_VIEW_VIDEO_PLAYBACK,
OnUpdateViewVideoPlayback)
    ON_COMMAND(ID_VIEW_CONFIGURATION, OnViewConfiguration)
    ON_COMMAND(ID_FILE_REGISTER_WITH_YELLOW_PAGES,
OnFileRegisterWithYellowPages)
    ON_COMMAND(ID_FILE_UNREGISTER_WITH_YELLOW_PAGES,
OnFileUnregisterWithYellowPages)
    ON_COMMAND(ID_HELP_EMAIL_SUPPORT, OnHelpEmailSupport)
    ON_COMMAND(ID_HELP_CHECK_FOR_UPDATES, OnHelpCheckForUpdates)
//}}AFX_MSG_MAP
// Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CProjectNalayApp construction

CProjectNalayApp::CProjectNalayApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
    gm_Layout.m_bConnectionsWindowOpen = FALSE;
    gm_Layout.m_bFeedbackWindowOpen = FALSE;
    gm_Layout.m_bPlaybackWindowOpen = FALSE;
    m_pFeedbackListView = NULL;
}

////////////////////////////////////
/////
// The one and only CProjectNalayApp object

CProjectNalayApp theApp;

void CProjectNalayApp::SplashScreen()
{
    CDialog t_Dlg;

    t_Dlg.Create(IDD_SPLASHSCREEN);
    Sleep(1000);
    t_Dlg.DestroyWindow();
}

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CProjectNalayApp initialization

BOOL CProjectNalayApp::InitInstance()
{
    HRESULT hr;
    CString t_str;
    char t_szAppDir[_MAX_PATH];

//    SplashScreen();

    // Get startup app directory
    lstrcpy ( t_szAppDir, m_pszHelpFilePath);
    PathRemoveFileSpec(t_szAppDir);
    gm_AppConfig.m_StartupPath = t_szAppDir;
//    _getcwd(m_szAppDir, _MAX_PATH);

    // Start with fresh Feedback file
    CString t_strFeedbackFilename;
    GetFeedbackFileName(t_strFeedbackFilename);
    DeleteFile(t_strFeedbackFilename);

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the
size
    // of your final executable, you should remove from the
following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();                // Call this when using MFC
in a shared DLL
#else
    Enable3dControlsStatic();          // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something
appropriate
    // such as the name of your company or organization.
    t_str.LoadString(IDS_COMPANY);
    SetRegistryKey(t_str);

    LoadStdProfileSettings(); // Load standard INI file options
(including MRU)

    // Load layout settings
    gm_Layout.LoadSettings();

    // Load config settings
    gm_AppConfig.LoadSettings();

```

```

// Check password
if ( !PromptSecurityPassword(PWDPROMPT_ONLAUNCH) )
    return FALSE;

// Register the application's document templates. Document
templates
// serve as the connection between documents, frame windows and
views.

CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_PROJECTYPE,
    RUNTIME_CLASS(CMainDoc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CMainView));
AddDocTemplate(pDocTemplate);

// create main MDI Frame window
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;

// Connections Window
m_pDocTemplateConnections = new CMultiDocTemplate(
    IDR_CONNECTIONS,
    RUNTIME_CLASS(CMainDoc),
    RUNTIME_CLASS(CConnectionsFrame),
    RUNTIME_CLASS(CConnectionsListView));
AddDocTemplate(m_pDocTemplateConnections);

// Feedback Window
m_pDocTemplateFeedback = new CMultiDocTemplate(
    IDR_FEEDBACK,
    RUNTIME_CLASS(CMainDoc),
    RUNTIME_CLASS(CFeedbackFrame),
    RUNTIME_CLASS(CFeedbackListView));
AddDocTemplate(m_pDocTemplateFeedback );

// Instant Messenger Window
m_pDocTemplateInstantMessenger = new CMultiDocTemplate(
    IDR_INSTANTMESSENGER,
    RUNTIME_CLASS(CInstantMessengerDoc),
    RUNTIME_CLASS(CInstantMessengerFrame),
    RUNTIME_CLASS(CInstantMessengerConversationView));
AddDocTemplate(m_pDocTemplateInstantMessenger );

// Local Video Window
m_pDocTemplateLocalVideo = new CMultiDocTemplate(
    IDR_LOCALVIDEOSURVEILLANCE,
    RUNTIME_CLASS(CLocalVideoDoc),
    RUNTIME_CLASS(CLocalVideoFrame),
    RUNTIME_CLASS(CLocalVideoEventsView));
AddDocTemplate(m_pDocTemplateLocalVideo );

// Video Playback

```

```

m_pDocTemplatePlayback = new CMultiDocTemplate(
    IDR_VIDEOPLAYBACK,
    RUNTIME_CLASS(CPlaybackDoc),
    RUNTIME_CLASS(CPlaybackFrame),
    RUNTIME_CLASS(CPlaybackListView));
AddDocTemplate(m_pDocTemplatePlayback );

// Remote Video Window
m_pDocTemplateRemoteVideo = new CMultiDocTemplate(
    IDR_REMOTEVIDEOSURVEILLANCE,
    RUNTIME_CLASS(CLocalVideoDoc),
    RUNTIME_CLASS(CRemoteVideoFrame),
    RUNTIME_CLASS(CRemoteVideoEventsView));
AddDocTemplate(m_pDocTemplateRemoteVideo );

// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// The main window has been initialized, so show and update it.
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();

// Load Connections data
gm_Connections.LoadSettings();

// Init COM so we can use CoCreateInstance
// Note: COINIT_MULTITHREADED interferes with MAPI
// which seems to need Apartment Threading for Simple MAPI (see
CSimpleMAPI class)
// hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED );
NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_DEBUG, "CoInitializeEx",
hr);
if ( FAILED( hr ) )
{
    NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
"CoInitializeEx", hr);
    NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_ERROR,
"CoInitializeEx", IDS_COMNOTFOUND);
    NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_ERROR,
"CoInitializeEx", IDS_SHUTDOWN);
}

// Load the layout
// This must follow CoInitialize, since CFeedbackVideoView and
other Windows might
// need COM capability
LoadLayout();

// Enables the ability to use critical sections
InitializeCriticalSection(&gm_csHostList);
InitializeCriticalSection(&gm_csEmailAction);

```

```

InitializeCriticalSection(&gm_csAudioAction);
InitializeCriticalSection(&gm_csPhoneAction);
InitializeCriticalSection(&gm_csX10Action);
InitializeCriticalSection(&gm_csRecordVideo);

    // Start out with Connections Window
    OpenConnectionsWindow();
    OpenFeedbackWindow();

    // Register with yellow pages if necessary
    if ( gm_AppConfig.m_AutoRegisterYellowPages )
        OnFileRegisterWithYellowPages();

    // Call Update if necessary
    if ( gm_AppConfig.m_CheckUpdateAtLaunch )
        OnHelpCheckForUpdates();

    return TRUE;
}

////////////////////////////////////
/////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

    // Dialog Data
    //{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    CString      m_CompanyName;
    CString      m_LegalCopyright;
    CString      m_ProductName;
    CString      m_ProductVersion;
    CString      m_IPAddress;
    //}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

    // Implementation
protected:
    //{AFX_MSG(CAboutDlg)
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{

```

```

    //{AFX_DATA_INIT(CAboutDlg)
    m_CompanyName = _T("");
    m_LegalCopyright = _T("");
    m_ProductName = _T("");
    m_ProductVersion = _T("");
    m_IPAddress = _T("");
    //}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CAboutDlg)
    DDX_Text(pDX, IDC_COMPANYNAME, m_CompanyName);
    DDX_Text(pDX, IDC_LEGALCOPYRIGHT, m_LegalCopyright);
    DDX_Text(pDX, IDC_PRODUCTNAME, m_ProductName);
    DDX_Text(pDX, IDC_PRODUCTVERSION, m_ProductVersion);
    DDX_Text(pDX, IDC_IPADDRESS, m_IPAddress);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{AFX_MSG_MAP(CAboutDlg)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CProjectNalayApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CProjectNalayApp message handlers

void CProjectNalayApp::NUpdateAllViews( CView* pSender, LPARAM lHint ,
CObject* pHint )
{
    POSITION t_curTemplatePos = GetFirstDocTemplatePosition();

    // Cycle through templates
    AfxLockTempMaps();
    while(t_curTemplatePos != NULL)
    {
        CDocTemplate* t_curTemplate =
GetNextDocTemplate(t_curTemplatePos);
        POSITION t_curDocPos = t_curTemplate-
>GetFirstDocPosition();

        // Cycle through templates to find matching one
        while(t_curDocPos != NULL)
        {
            CMainDoc* t_Document = (CMainDoc* ) t_curTemplate-
>GetNextDoc(t_curDocPos);

```

```

        t_Document->UpdateAllViews( pSender, lHint , pHint );
    }
    AfxUnlockTempMaps();
}

void CProjectNalayApp::OnViewConnections()
{
    OpenConnectionsWindow();
}

void CProjectNalayApp::OnUpdateViewConnections(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_Layout.m_bConnectionsWindowOpen );
}

void CProjectNalayApp::OnViewFeedback()
{
    OpenFeedbackWindow();
}

void CProjectNalayApp::OnUpdateViewFeedback(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_Layout.m_bFeedbackWindowOpen );
}

BOOL CAboutDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    BOOL    bRes;
    char    szFilename[MAX_PATH];
    DWORD   dwVerInfoSize;           // Size of version
information block
    DWORD   dwVerHnd    = 0;         // An 'ignored' parameter,
always '0'
    HANDLE  hMem        = 0;
    LPSTR   pszVerInfo  = 0;         // Pointer to version
information block
    LPSTR   pszField;               // Pointer to
particular field within block
    UINT    uVerLen;               // Size of particular
field string returned
    NVersionInfo t_nVersionInfo;

    bRes = FALSE;    // Assume we'll fail until we know better

    // Get the IOP file path and name

    if (!GetModuleFileName(AfxGetApp()->m_hInstance, szFilename,
MAX_PATH))
        goto end;

    // Get the version info

    dwVerInfoSize = GetFileVersionInfoSize(szFilename, &dwVerHnd);

```

```

    if (dwVerInfoSize == 0) goto nd;

    hMem = GlobalAlloc(GMEM_MOVEABLE, dwVerInfoSiz );
    if (hMem == NULL) goto end;

    pszVerInfo = (char*) GlobalLock(hMem);
    if (pszVerInfo == NULL) goto end;

    GetFileVersionInfo(szFilename, dwVerHnd, dwVerInfoSize,
    pszVerInfo);

    // Fill the values

    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\CompanyName", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strCompanyName = pszField;
    m_CompanyName = t_nVersionInfo.strCompanyName = pszField;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\FileDescription", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strFileDescription = pszField;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\FileVersion", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strFileVersion = pszField;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\Internal Name", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strInternalName = pszField;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\LegalCopyright", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strLegalCopyright = pszField;
    m_LegalCopyright = t_nVersionInfo.strLegalCopyright;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\OriginalFilename", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strOriginalFilename = pszField;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\ProductName", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strProductName = pszField;
    m_ProductName = t_nVersionInfo.strProductName ;
    pszField = 0;
    VerQueryValue((LPVOID) pszVerInfo,
    "\\StringFileInfo\\040904B0\\ProductVersion", (LPVOID*) &pszField,
    &uVerLen);
    t_nVersionInfo.strProductVersion = pszField;
    m_ProductVersion .LoadString(IDS_VERSION);

```

```

        m_ProductVersion += t_nVersionInfo.strProductVersion ;
        pszField = 0;
        VerQueryValue((LPVOID) pszVerInfo,
"\StringFileInfo\040904B0\Comments", (LPVOID*) &pszField, &uVerLen);
        t_nVersionInfo.strComments = pszField;
        pszField = 0;
        VerQueryValue((LPVOID) pszVerInfo,
"\StringFileInfo\040904B0\LegalTrademarks", (LPVOID*) &pszField,
&uVerLen);
        t_nVersionInfo.strLegalTrademarks = pszField;
        pszField = 0;
        VerQueryValue((LPVOID) pszVerInfo,
"\StringFileInfo\040904B0\PrivateBuild", (LPVOID*) &pszField,
&uVerLen);
        t_nVersionInfo.strPrivateBuild = pszField;
        pszField = 0;
        VerQueryValue((LPVOID) pszVerInfo,
"\StringFileInfo\040904B0\SpecialBuild", (LPVOID*) &pszField,
&uVerLen);
        t_nVersionInfo.strSpecialBuild = pszField;

        m_IPAddress.LoadString(IDS_CURRENTIPADDRESS);
        m_IPAddress += GetCurrentIPAddress();

        UpdateData(FALSE);

        bRes = TRUE;
end:
        // Cleanup version data
        if (hMem)
        {
                if (pszVerInfo) GlobalUnlock(hMem);
                GlobalFree(hMem);
        }

        return TRUE; // return TRUE unless you set the focus to a
control
        // EXCEPTION: OCX Property Pages should return
FALSE
}

int CProjectNalayApp::ExitInstance()
{
        // Save Connections data
        gm_Connections.SaveSettings();

        // Save layout information
        gm_Layout.SaveSettings();

        // Save config information
        gm_AppConfig.SaveSettings();

        // Delete critical section object
        DeleteCriticalSection(&gm_csHostList);
        DeleteCriticalSection(&gm_csEMailAction);
        DeleteCriticalSection(&gm_csAudioAction);

```

```

DeleteCriticalSection(&gm_csPhoneAction);
DeleteCriticalSection(&gm_csX10Action);

    // Uninitialize COM
    CoUninitialize();

    return CWinApp::ExitInstance();
}

bool CProjectNalayApp::OpenConnectionWindow(CString strLabel, bool
bNewLayout)
{
    bool t_bReturn = false;
    UINT t_nConnectionType;
    UINT t_nConnectionMethod;

    // See if the Connections Window is already open
    if ( bNewLayout && IsConnectionWindowOpen(strLabel) )
        goto Exit1;

    // Get the connection type
    if ( !gm_Connections.GetConnectionType(strLabel,
t_nConnectionType) )
        goto Exit1;

    // Get the connection method
    if ( !gm_Connections.GetConnectionMethod(strLabel,
t_nConnectionMethod) )
        goto Exit1;

    // Open the connection window
    switch ( t_nConnectionType )
    {
    case CONTYPE_INSTANTMESSENGER:
        if ( !OpenInstantMessengerWindow(strLabel) )
            break;
        t_bReturn = true;
        break;
    case CONTYPE_VIDEOSURVEILLANCE:
        switch ( t_nConnectionMethod )
        {
        case CONNECTION_LOCALTCPIP:
            if ( !OpenLocalVideoWindow(strLabel) )
                break;
            t_bReturn = true;
            break;
        case CONNECTION_IPADDRESS:
        case CONNECTION_YELLOWPAGES:
            if ( !OpenRemoteVideoWindow(strLabel) )
                break;
            t_bReturn = true;
            break;
        };
        break;
    }

    // Update layout manager

```

```

        if ( t_bReturn && bNewLayout)
            gm_Layout.AddLayoutConnectionWindow(strLabel);

Exit1:
    return t_bReturn ;
}

bool CProjectNalayApp::CloseConnectionWindow(CString strLabel)
{
    CString t_strLabel = strLabel;

    NUpdateAllViews(NULL, NUPDATE_CLOSECONNECTIONWINDOW, (Object*)
&t_strLabel);

    return true;
}

void CProjectNalayApp::OnViewVideoPlayback()
{
    OpenPlaybackWindow();
}

void CProjectNalayApp::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // storing code
    }
    else
    {
        // loading code
    }
}

bool CProjectNalayApp::OpenLocalVideoWindow(CString strConnectionLabel)
{
    bool t_bReturn = false;
    CString t_strTitle, t_str;
    t_str.LoadString(IDS_LOCALVIDEOSURVEILLANCE);

    t_strTitle.Format("%s: %s", (LPCTSTR) t_str, (LPCTSTR)
strConnectionLabel);

    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
m_pDocTemplateLocalVideo->OpenDocumentFile(NULL);
    if ( t_LocalVideoDoc != NULL )
    {
        t_LocalVideoDoc->m_ConnectionLabel = strConnectionLabel;
        t_LocalVideoDoc->SetTitle(t_strTitle);
        AfxGetMainWnd()->UpdateWindow();
        t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_REFRESH,
NULL);

        t_bReturn = true;
    }

    return t_bReturn;
}

```

```

bool CProjectNalayApp::OpenInstantMessengerWindow(CString
strConnectionLabel)
{
    bool t_bReturn = false;
    CString t_strTitle, t_str;
    t_str.LoadString(IDS_INSTANTMESSENGER);

    t_strTitle.Format("%s: %s", (LPCTSTR) t_str, (LPCTSTR)
strConnectionLabel);

    CInstantMessengerDoc * t_InstantMessengerDoc =
(CInstantMessengerDoc * ) m_pDocTemplateInstantMessenger-
>OpenDocumentFile(NULL);
    if ( t_InstantMessengerDoc != NULL )
    {
        t_InstantMessengerDoc ->m_ConnectionLabel =
strConnectionLabel;
        t_InstantMessengerDoc ->SetTitle(t_strTitle);
        AfxGetMainWnd()->UpdateWindow();
        t_InstantMessengerDoc->UpdateAllViews(NULL,
NUPDATE_REFRESH, NULL);
        t_bReturn = true;
    }

    return t_bReturn;
}

bool CProjectNalayApp::OpenRemoteVideoWindow(CString
strConnectionLabel)
{
    bool t_bReturn = false;
    CString t_strTitle, t_str;
    t_str.LoadString(IDS_REMOTEVIDEOSURVEILLANCE);

    t_strTitle.Format("%s: %s", (LPCTSTR) t_str, (LPCTSTR)
strConnectionLabel);

    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
m_pDocTemplateRemoteVideo->OpenDocumentFile(NULL);
    if ( t_LocalVideoDoc != NULL )
    {
        t_LocalVideoDoc->m_ConnectionLabel = strConnectionLabel;
        t_LocalVideoDoc->SetTitle(t_strTitle);
        AfxGetMainWnd()->UpdateWindow();
        t_LocalVideoDoc->UpdateAllViews(NULL, NUPDATE_REFRESH,
NULL);

        t_bReturn = true;
    }

    return t_bReturn;
}

bool CProjectNalayApp::OpenConnectionsWindow()
{
    bool t_bReturn = false;
    CString t_strTitle;

```

```

CMainDoc* t_MainDoc ;

// If Connections window is open already do not reopen
if ( gm_Layout.m_bConnectionsWindowOpen )
    goto Exit1;

// Get doc handle and open a new Connections window
t_MainDoc = (CMainDoc * ) m_pDocTemplateConnections-
>OpenDocumentFile(NULL);
if ( t_MainDoc != NULL )
{
    // Update window - especially needed if opening before main
    frame is visible
    AfxGetMainWnd()->UpdateWindow();

    // Set title name
    t_strTitle.LoadString(IDS_CONNECTIONS);
    t_MainDoc->SetTitle(t_strTitle);

    // Update variable to ensure window will not open twice
    gm_Layout.m_bConnectionsWindowOpen = TRUE;
    t_bReturn = true;
}

Exit1:
    return t_bReturn;
}

CLayout::CLayout()
{
    // Initialize values
    m_rectMainWindow.SetRect(50, 50, 700, 570);

    m_rectConnectionsWindow.SetRect(0, 0, 502, 150);
    m_bConnectionsWindowOpen = FALSE;

    m_rectFeedbackWindow.SetRect(0, 150, 502, 400);
    m_bFeedbackWindowOpen = FALSE;

    m_rectPlaybackWindow.SetRect(50, 50, 625, 350);
    m_bPlaybackWindowOpen = FALSE;
}

void CLayout::SaveSettings()
{
    CString strFileName;
    CFile t_File;

    GetSerializeFileName(strFileName);

    if ( !t_File.Open((LPCTSTR) strFileName, CFile::modeWrite |
CFile::modeCreate ) )
        return;

    CArchive t_archive(&t_File, CArchive::store);

    Serialize( t_archive );
}

```

```

        t_archive.Close();
        t_File.Close();
    }

bool CLayout::GetSerializeFileName(CString& strFileName)
{
    CString t_strDefaultFilename;
    char t_szSavePath[_MAX_PATH];

    lstrcpy (t_szSavePath, gm_AppConfig.m_StartupPath);

    // Create save path
    t_strDefaultFilename.LoadString(IDS_DEFAULTLAYOUTFILENAME);
    PathAppend(t_szSavePath, t_strDefaultFilename);

    strFileName = t_szSavePath;
    return true;
}

void CLayout::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
    {
        archive << m_rectMainWindow;

        archive << m_bConnectionsWindowOpen;
        archive << m_rectConnectionsWindow;

        archive << m_bFeedbackWindowOpen;
        archive << m_rectFeedbackWindow;

        archive << m_bPlaybackWindowOpen;
        archive << m_rectPlaybackWindow;

        // archive Connection windows
        archive << m_LayoutConnections.GetSize();

        for ( int i = 0; i < m_LayoutConnections.GetSize(); i++ )
            m_LayoutConnections.ElementAt(i).Serialize( archive );
    };
    else
    {
        int t_nLayoutConnections;
        CLayoutConnectionWindow t_LayoutConnectionWindow;

        // Clear all connection window layout data
        m_LayoutConnections.RemoveAll();

        archive >> m_rectMainWindow;
    }
}

```

```

archive >> m_bConnectionsWindowOpen;
    archive >> m_rectConnectionsWindow;

archive >> m_bFeedbackWindowOpen;
    archive >> m_rectFeedbackWindow;

archive >> m_bPlaybackWindowOpen;
    archive >> m_rectPlaybackWindow;

    // archive Connection windows
    archive >> t_nLayoutConnections;

    for ( int i = 0; i < t_nLayoutConnections; i++ )
    {
        t_LayoutConnectionWindow.Serialize( archive );
        m_LayoutConnections.Add(t_LayoutConnectionWindow);
    }
}

}

bool CProjectNalayApp::LoadLayout()
{
    // Show connections window where it left off
    if ( gm_Layout.m_bConnectionsWindowOpen )
    {
        // Insurance in case something fails
        gm_Layout.m_bConnectionsWindowOpen = FALSE;
        OpenConnectionsWindow();
    }

    // Show feedback window where it left off
    if ( gm_Layout.m_bFeedbackWindowOpen )
    {
        // Insurance in case something fails
        gm_Layout.m_bFeedbackWindowOpen = FALSE;
        OpenFeedbackWindow();
    }

    // Show video playback window where it left off
    if ( gm_Layout.m_bPlaybackWindowOpen )
    {
        // Insurance in case something fails
        gm_Layout.m_bPlaybackWindowOpen = FALSE;
        OpenPlaybackWindow();
    }

    // Open Connection windows as they were
    for ( int i = 0; i < gm_Layout.m_LayoutConnections.GetSize(); i++
)

        OpenConnectionWindow(gm_Layout.m_LayoutConnections[i].m_Label,
false);

    return true;
}

```

```

bool CProjectNalayApp::OpenFeedbackWindow()
{
    bool t_bReturn = false;
    CString t_strTitle;
    CMainDoc* t_MainDoc ;

    // If Feedback window is open already do not reopen
    if ( gm_Layout.m_bFeedbackWindowOpen )
        goto Exit1;

    // Get doc handle and open a new Feedback window
    t_MainDoc = (CMainDoc * ) m_pDocTemplateFeedback-
>OpenDocumentFile(NULL);
    if ( t_MainDoc != NULL )
    {
        // Update window - especially needed if opening before main
frame is visible
        AfxGetMainWnd()->UpdateWindow();

        // Set title name
        t_strTitle.LoadString(IDS_FEEDBACK);
        t_MainDoc->SetTitle(t_strTitle);

        // Set feedback view
        SetFeedbackListView( t_MainDoc );

        // Update variable to ensure window will not open twice
        gm_Layout.m_bFeedbackWindowOpen = TRUE;
        t_bReturn = true;
    }

Exit1:
    return t_bReturn;
}

void CProjectNalayApp::CloseFeedbackListView( )
{
    m_pFeedbackListView = NULL;
}

bool CProjectNalayApp::SetFeedbackListView( CMainDoc * MainDoc )
{
    bool t_bReturn = false;

    POSITION t_pos = MainDoc->GetFirstViewPosition();

    if ( t_pos != NULL )
    {
        m_pFeedbackListView = ( CFeedbackListView * ) MainDoc-
>GetNextView( t_pos );
        t_bReturn = true;
    }

    return t_bReturn;
}

```

```

bool CProjectNalayApp::OpenPlaybackWindow()
{
    bool t_bReturn = false;
    CString t_strTitle;
    CMainDoc* t_MainDoc ;

    // If Feedback window is open already do not reopen
    if ( gm_Layout.m_bPlaybackWindowOpen )
        goto Exit1;

    // Get doc handle and open a new Feedback window
    t_MainDoc = (CMainDoc * ) m_pDocTemplatePlayback-
>OpenDocumentFile(NULL);
    if ( t_MainDoc != NULL )
    {
        // Update window - especially needed if opening before main
frame is visible
        AfxGetMainWnd()->UpdateWindow();

        // Set title name
        t_strTitle.LoadString(IDS_VIDEOPLAYBACK);
        t_MainDoc->SetTitle(t_strTitle);

        // Update variable to ensure window will not open twice
        gm_Layout.m_bPlaybackWindowOpen = TRUE;
        t_bReturn = true;
    }

Exit1:
    return t_bReturn;
}

CLayoutConnectionWindow::CLayoutConnectionWindow()
{
    m_rectWindow.SetRect(25, 25, 400, 400);
}

void CLayoutConnectionWindow::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
    {
        archive << m_Label;
        archive << m_rectWindow;
    }
    else
    {
        archive >> m_Label;
        archive >> m_rectWindow;
    }
}

bool CLayout::AddLayoutConnectionWindow(CString strConnectionLabel)

```

```

{
    CLayoutConnectionWindow t_LayoutConnectionWindow;

    t_LayoutConnectionWindow.m_Label = strConnectionLabel;

    m_LayoutConnections.Add(t_LayoutConnectionWindow);

    return true;
}

CLayoutConnectionWindow&
CLayoutConnectionWindow::operator=(CLayoutConnectionWindow&
LayoutConnectionWindow)
{
    m_Label = LayoutConnectionWindow.m_Label;
    m_rectWindow = LayoutConnectionWindow.m_rectWindow;
    return LayoutConnectionWindow;
}

bool CLayout::DeleteLayoutConnection(CString strConnectionLabel)
{
    bool t_bReturn = false;
    int t_nIndex;

    t_nIndex = GetLayoutConnectionIndexFromLabel(strConnectionLabel);

    if ( t_nIndex == -1 )
        goto Exit1;

    m_LayoutConnections.RemoveAt(t_nIndex);

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

int CLayout::GetLayoutConnectionIndexFromLabel(CString strLabel)
{
    int t_nIndex = -1;

    // Search through connections until there is a label match
    for (int i = 0; i < m_LayoutConnections.GetSize(); i++ )
    {
        if ( m_LayoutConnections.ElementAt(i).m_Label == strLabel )
            t_nIndex = i;
    }

    return t_nIndex;
}

void CProjectNalayApp::OnUpdateViewVideoPlayback(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(gm_Layout.m_bPlaybackWindowOpen );
}

void CProjectNalayApp::OnViewConfiguration()

```

```

{
    CString t_str;

    // Check password
    if ( !PromptSecurityPassword(PWDPROMPT_ONCONFIGURATION) )
        return ;

    // Create a blank property sheet
    t_str.LoadString(IDS_CONFIGURATION);

    CPropertySheet t_PS(t_str);
    t_PS.m_psh.dwFlags |= PSP_USEHICON;
    HICON t_hIcon = LoadIcon(MAKEINTRESOURCE(IDR_MAINFRAME));
    t_PS.m_psh.dwFlags |= PSH_NOAPPLYNOW;
    t_PS.m_psh.hIcon = t_hIcon;

    // Create pages and initialize values
    CPropPageEmail t_PropPageEmail;

    CPropPageConnections t_PropPageConnections;
    t_PropPageConnections.m_OnDoubleClick =
gm_AppConfig.m_nOnDoubleClickConnectionsItem;

    CPropPageFeedback t_PropPageFeedback;
    t_PropPageFeedback.m_ShowDebugFeedback =
gm_AppConfig.m_ShowDebugFeedback;
    t_PropPageFeedback.m_ShowErrorFeedback =
gm_AppConfig.m_ShowErrorFeedback ;
    t_PropPageFeedback.m_ShowWarningFeedback =
gm_AppConfig.m_ShowWarningFeedback;
    t_PropPageFeedback.m_ShowStatusFeedback =
gm_AppConfig.m_ShowStatusFeedback ;

    CPropPagePhoneModem t_PropPagePhoneModem;
    t_PropPagePhoneModem.m_TAPIDevice = gm_AppConfig.m_TAPIDevice;

    CPropPageRecordVideo t_PropPageRecordVideo;
    t_PropPageRecordVideo.m_DefaultRecordDuration =
gm_AppConfig.m_DefaultVideoRecordDuration;
    t_PropPageRecordVideo.m_DefaultRecordFile =
gm_AppConfig.m_DefaultVideoRecordFilename;
    t_PropPageRecordVideo.m_MaxContinuousFiles =
gm_AppConfig.m_MaxContinuousFiles;

    CPropPageVideoPlayback t_PropPageVideoPlayback;
    t_PropPageVideoPlayback.m_ForwardSpeed =
gm_AppConfig.m_ForwardSpeed;
    t_PropPageVideoPlayback.m_FastForwardSpeed =
gm_AppConfig.m_FastForwardSpeed;
    t_PropPageVideoPlayback.m_DefaultDirectory =
gm_AppConfig.m_DefaultVideoPlaybackDirectory;

    CPropPageYellowPages t_PropPageYellowPages;
    t_PropPageYellowPages.m_AutoRegisterYellowPages =
gm_AppConfig.m_AutoRegisterYellowPages;

    CPropPageVideo t_PropPageVideo;

```

```

        CPropPageAudio t_PropPageAudio;
        t_PropPageAudio.m_DefaultDirectory =
gm_AppConfig.m_DefaultAudioDirectory;

        CPropPageSecurityPrompts t_PropPageSecurityPrompts;
        t_PropPageSecurityPrompts.m_PromptOnExit =
gm_AppConfig.m_PromptOnExit ;
        t_PropPageSecurityPrompts.m_EnableAppSecurityPrompt =
gm_AppConfig.m_EnableAppSecurityPrompt ;
        t_PropPageSecurityPrompts.m_AppPassword =
gm_AppConfig.m_AppPassword ;
        t_PropPageSecurityPrompts.m_AppPassword2 =
gm_AppConfig.m_AppPassword ;
        t_PropPageSecurityPrompts.m_PwdPromptOnLaunch =
gm_AppConfig.m_PwdPromptOnLaunch ;
        t_PropPageSecurityPrompts.m_PwdPromptOnExit =
gm_AppConfig.m_PwdPromptOnExit ;
        t_PropPageSecurityPrompts.m_PwdPromptOnAlarm =
gm_AppConfig.m_PwdPromptOnAlarm ;
        t_PropPageSecurityPrompts.m_PwdPromptOnConnection =
gm_AppConfig.m_PwdPromptOnConnection ;
        t_PropPageSecurityPrompts.m_PwdPromptOnConfiguration =
gm_AppConfig.m_PwdPromptOnConfiguration ;

        CPropPageUpdate t_PropPageUpdate;
        t_PropPageUpdate.m_CheckForUpdateAtLaunch =
gm_AppConfig.m_CheckUpdateAtLaunch;

        // Add property pages
        t_PS.AddPage(&t_PropPageSecurityPrompts);
        t_PS.AddPage(&t_PropPageConnections);
        t_PS.AddPage(&t_PropPageVideoPlayback);
        t_PS.AddPage(&t_PropPageFeedback);
//      t_PS.AddPage(&t_PropPageEMail);
        t_PS.AddPage(&t_PropPagePhoneModem);
        t_PS.AddPage(&t_PropPageRecordVideo);
        t_PS.AddPage(&t_PropPageYellowPages);
        t_PS.AddPage(&t_PropPageVideo);
        t_PS.AddPage(&t_PropPageAudio);
        t_PS.AddPage(&t_PropPageUpdate);

        // Need to lock the maps - fix for a bug in MFC
        AfxLockTempMaps();
        if ( t_PS.DoModal() == IDOK )
        {
            // Get property page values

            // Connection Data
            gm_AppConfig.m_OnDoubleClickConnectionsItem =
t_PropPageConnections.m_OnDoubleClick ;

            // Feedback
            gm_AppConfig.m_ShowDebugFeedback =
t_PropPageFeedback.m_ShowDebugFeedback ;
            gm_AppConfig.m_ShowErrorFeedback =
t_PropPageFeedback.m_ShowErrorFeedback ;

```

```

        gm_AppConfig.m_ShowWarningFeedback =
t_PropPageFeedback.m_ShowWarningFeedback ;
        gm_AppConfig.m_ShowStatusFeedback =
t_PropPageFeedback.m_ShowStatusFeedback ;

        // TAPI
        gm_AppConfig.m_TAPIDevice =
t_PropPagePhoneModem.m_TAPIDevice ;

        // Video Record
        gm_AppConfig.m_DefaultVideoRecordDuration =
t_PropPageRecordVideo.m_DefaultRecordDuration ;
        gm_AppConfig.m_DefaultVideoRecordFilename =
t_PropPageRecordVideo.m_DefaultRecordFile ;
        gm_AppConfig.m_MaxContinuousFiles =
t_PropPageRecordVideo.m_MaxContinuousFiles ;

        // Video Playback
        gm_AppConfig.m_ForwardSpeed =
t_PropPageVideoPlayback.m_ForwardSpeed ;
        gm_AppConfig.m_FastForwardSpeed =
t_PropPageVideoPlayback.m_FastForwardSpeed ;
        gm_AppConfig.m_DefaultVideoPlaybackDirectory =
t_PropPageVideoPlayback.m_DefaultDirectory ;

        // Yellow Pages
        gm_AppConfig.m_AutoRegisterYellowPages =
t_PropPageYellowPages.m_AutoRegisterYellowPages ;

        // Security Prompts
        gm_AppConfig.m_PromptOnExit =
t_PropPageSecurityPrompts.m_PromptOnExit ;
        gm_AppConfig.m_EnableAppSecurityPrompt =
t_PropPageSecurityPrompts.m_EnableAppSecurityPrompt ;
        gm_AppConfig.m_AppPassword =
t_PropPageSecurityPrompts.m_AppPassword ;
        gm_AppConfig.m_PwdPromptOnLaunch =
t_PropPageSecurityPrompts.m_PwdPromptOnLaunch ;
        gm_AppConfig.m_PwdPromptOnExit =
t_PropPageSecurityPrompts.m_PwdPromptOnExit ;
        gm_AppConfig.m_PwdPromptOnAlarm =
t_PropPageSecurityPrompts.m_PwdPromptOnAlarm ;
        gm_AppConfig.m_PwdPromptOnConnection =
t_PropPageSecurityPrompts.m_PwdPromptOnConnection ;
        gm_AppConfig.m_PwdPromptOnConfiguration =
t_PropPageSecurityPrompts.m_PwdPromptOnConfiguration ;

        // Audio
        gm_AppConfig.m_DefaultAudioDirectory =
t_PropPageAudio.m_DefaultDirectory ;

        // Update
        gm_AppConfig.m_CheckUpdateAtLaunch =
t_PropPageUpdate.m_CheckForUpdateAtLaunch ;
    }
    AfxUnlockTempMaps();
}

```

```

CAppConfig::CAppConfig()
{
    m_nOnDoubleClickConnectionsItem =
ONDBLCLICKCONNECTIONITEM_OPENCONNECTION;

    // Feedback
    m_ShowDebugFeedback = FALSE;
    m_ShowErrorFeedback= TRUE;
    m_ShowStatusFeedback= FALSE;
    m_ShowWarningFeedback= TRUE;

    // Video Record
    m_DefaultVideoRecordDuration = CTime(2000, 1, 1, 0, 0, 20);
    m_MaxContinuousFiles = 2;

    // Video Playback
    m_ForwardSpeed = 2;
    m_FastForwardSpeed = 4;
//
    m_DefaultVideoPlaybackDirectory.LoadString(IDS_MYDOCUMENTSDIRECTO
RY);

    // Yellow Pages
    m_AutoRegisterYellowPages = FALSE;

    // Security Prompts
    m_PromptOnExit = TRUE;
    m_EnableAppSecurityPrompt = FALSE;
    m_PwdPromptOnLaunch = FALSE;
    m_PwdPromptOnExit = FALSE;
    m_PwdPromptOnAlarm = FALSE;
    m_PwdPromptOnConnection = FALSE;
    m_PwdPromptOnConfiguration = FALSE;

    // Update
    m_CheckUpdateAtLaunch = TRUE;
}

#define PROJNAL_LAYOUTFILETIME CTime(2002, 4, 17, 2, 33, 0)

bool CLayout::LoadSettings()
{
    CString t_strDefaultFilename;
    CFile t_File;
    CFileStatus t_FileStatus;

    GetSerializeFileName(t_strDefaultFilename);
    if ( !t_File.Open(t_strDefaultFilename, CFile::modeRead) )
        return false;

    // Make sure we are not reading an old file
    t_File.GetStatus(t_FileStatus);
    if ( t_FileStatus.m_mtime > PROJNAL_LAYOUTFILETIME )
    {
        CArchive t_archive(&t_File, CArchive::load);
        Serialize( t_archive );
    }
}

```

```

        t_archive.Close();
    }
    t_File.Close();

    return true;
}

#define PROJNAL_APPCONFIGFILETIME CTime(2002, 6, 27, 17, 05, 0)

bool CAppConfig::LoadSettings()
{
    CString t_strDefaultFilename;
    CFile t_File;
    CFileStatus t_FileStatus;

    GetSerializeFileName(t_strDefaultFilename);
    if ( !t_File.Open(t_strDefaultFilename, CFile::modeRead) )
        return false;

    // Make sure we are not reading an old file
    t_File.GetStatus(t_FileStatus);
    if ( t_FileStatus.m_mtime > PROJNAL_APPCONFIGFILETIME )
    {
        CArchive t_archive(&t_File, CArchive::load);
        Serialize( t_archive );
        t_archive.Close();
    }
    t_File.Close();

    // Apply app directory to empty default directories
    if ( m_DefaultAudioDirectory.IsEmpty() )
        m_DefaultAudioDirectory = m_StartupPath ;

    if ( m_DefaultVideoPlaybackDirectory.IsEmpty() )
        m_DefaultVideoPlaybackDirectory = m_StartupPath ;

    return true;
}

void CAppConfig::SaveSettings()
{
    CString strFileName;
    CFile t_File;

    GetSerializeFileName(strFileName);

    if ( !t_File.Open((LPCTSTR) strFileName, CFile::modeWrite |
CFile::modeCreate ) )
        return;

    CArchive t_archive(&t_File, CArchive::store);

    Serialize( t_archive );

    t_archive.Close();
    t_File.Close();
}

```

```

bool CAppConfig::GetSerializeFileName(CString& strFileName)
{
    CString t_strDefaultFilename;
    char t_szSavePath[_MAX_PATH];

    lstrcpy (t_szSavePath, m_StartupPath);

    // Create save path
    t_strDefaultFilename.LoadString(IDS_DEFAULTCONFIGFILENAME);
    PathAppend(t_szSavePath, t_strDefaultFilename);

    strFileName = t_szSavePath;
    return true;
}

void CAppConfig::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
    {
        // Connections data
        archive << m_nOnDoubleClickConnectionsItem;

        // Feedback
        archive << m_ShowDebugFeedback;
        archive << m_ShowErrorFeedback;
        archive << m_ShowStatusFeedback;
        archive << m_ShowWarningFeedback;

        // TAPI
        archive << m_TAPIDevice;

        // Video Record
        archive << m_DefaultVideoRecordDuration;
        archive << m_DefaultVideoRecordFilename;
        archive << m_MaxContinuousFiles;

        // Video Playback
        archive << m_ForwardSpeed;
        archive << m_FastForwardSpeed;
        archive << m_DefaultVideoPlaybackDirectory;

        // Yellow Pages
        archive << m_AutoRegisterYellowPages;

        // Security Prompts
        archive << m_PromptOnExit;
        archive << m_EnableAppSecurityPrompt;
        archive << m_AppPassword;
        archive << m_PwdPromptOnLaunch;
        archive << m_PwdPromptOnExit;
        archive << m_PwdPromptOnAlarm;
    }
}

```

```

        archive << m_PwdPromptOnConnection;
        archive << m_PwdPromptOnConfiguration;

        // Audio
        archive << m_DefaultAudioDirectory;

        // Update
        archive << m_CheckUpdateAtLaunch;;
    }
else
{
    // Connections data
    archive >> m_nOnDoubleClickConnectionsItem;

    // Feedback
    archive >> m_ShowDebugFeedback;
    archive >> m_ShowErrorFeedback;
    archive >> m_ShowStatusFeedback;
    archive >> m_ShowWarningFeedback;

    // TAPI
    archive >> m_TAPIDevice;

    // Video Record
    archive >> m_DefaultVideoRecordDuration;
    archive >> m_DefaultVideoRecordFilename;
    archive >> m_MaxContinuousFiles;

    // Video Playback
    archive >> m_ForwardSpeed;
    archive >> m_FastForwardSpeed;
    archive >> m_DefaultVideoPlaybackDirectory;

    // Yellow Pages
    archive >> m_AutoRegisterYellowPages;

    // Security Prompts
    archive >> m_PromptOnExit;
    archive >> m_EnableAppSecurityPrompt;
    archive >> m_AppPassword;
    archive >> m_PwdPromptOnLaunch;
    archive >> m_PwdPromptOnExit;
    archive >> m_PwdPromptOnAlarm;
    archive >> m_PwdPromptOnConnection;
    archive >> m_PwdPromptOnConfiguration;

    // Audio
    archive >> m_DefaultAudioDirectory;

    // Update
    archive >> m_CheckUpdateAtLaunch;;
}

}

bool CProjectNalayApp::IsConnectionWindowOpen(CString
t_strConnectionLabel)
{

```

```

        bool t_bReturn = false;

        for ( int i = 0; i < gm_Layout.m_LayoutConnections.GetSize() ;
i++ )
        {
            if ( gm_Layout.m_LayoutConnections[i].m_Label ==
t_strConnectionLabel )
            {
                t_bReturn = true;
                break;
            }
        }

        return t_bReturn;
    }

bool CLayout::GetConnectionWindowRect(CString strConnectionLabel, CRect
& RectWindow )
{
    bool t_bReturn = false;
    int t_nIndex;

    t_nIndex = GetLayoutConnectionIndexFromLabel(strConnectionLabel);

    if ( t_nIndex == -1 )
        goto Exit1;

    RectWindow = m_LayoutConnections.ElementAt(t_nIndex).m_rectWindow
;

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

bool CLayout::SetConnectionWindowRect(CString strConnectionLabel, CRect
RectWindow )
{
    bool t_bReturn = false;
    int t_nIndex;

    t_nIndex = GetLayoutConnectionIndexFromLabel(strConnectionLabel);

    if ( t_nIndex == -1 )
        goto Exit1;

    m_LayoutConnections.ElementAt(t_nIndex).m_rectWindow = RectWindow
;

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

#include "atlbase.h"
#import "..\Yellow Pages\VPD.ocx" named_guids no_namespace
void CProjectNalayApp::OnFileRegisterWithYellowPages()

```

```

{
    CComPtr<IVideoPeer> t_pYellowPages;
    CComBSTR t_bstrLabel;
    CComBSTR t_bstrPassword;
    CComBSTR t_bstrIPAddress;
    VARIANT_BOOL t_bVisible = VARIANT_TRUE;
    VARIANT_BOOL t_bPrompt = VARIANT_TRUE;
    HRESULT t_hResult;

    t_hResult = t_pYellowPages.CoCreateInstance(CLSID_VideoPeer);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
            "OnFileRegisterWithYellowPages", t_hResult);
        goto Exit1;
    }

    t_hResult = t_pYellowPages->Register(t_bstrLabel.m_str,
        t_bstrPassword.m_str, t_bVisible, t_bPrompt, t_bstrIPAddress.m_str);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
            "OnFileRegisterWithYellowPages", t_hResult);
        goto Exit1;
    }

    // t_pYellowPages.Release();

Exit1:
    return ;
}

void CProjectNalayApp::OnFileUnregisterWithYellowPages()
{
    CComPtr<IVideoPeer> t_pYellowPages;
    CComBSTR t_bstrLabel;
    CComBSTR t_bstrPassword;
    VARIANT_BOOL t_bPrompt = VARIANT_TRUE;

    HRESULT t_hResult;

    t_hResult = t_pYellowPages.CoCreateInstance(CLSID_VideoPeer);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
            "OnFileUnregisterWithYellowPages", t_hResult);
        goto Exit1;
    }

    t_hResult = t_pYellowPages->Unregister(t_bstrLabel.m_str,
        t_bstrPassword.m_str, t_bPrompt);
    if ( FAILED(t_hResult) )
    {
        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
            "OnFileUnregisterWithYellowPages", t_hResult);
        goto Exit1;
    }
}

```

```

// t_pYellowPages.Release();

Exit1:
    return ;
}

void CProjectNalayApp::OnHelpEmailSupport()
{
    CSimpleMAPI t_SimpleMAPI;
    CString t_strSubject;
    CString t_strMessage;
    CString t_strAttachments;
    CString t_strLayoutAttachment;
    CString t_strConfigurationAttachment;
    CString t_strConnectionsAttachment;
    CString t_strFeedbackAttachment;
    CString t_strTo;

    // Save settings so we have something to send
    gm_Layout.SaveSettings();
    gm_Connections.SaveSettings();
    gm_AppConfig.SaveSettings();
    // Simple way of creating a feedback file to send
    NSENDFEEDBACKHRESULT(FEEDBACKMESSAGE_TYPE_DEBUG,
        "OnHelpEmailSupport", S_OK);

    // Get full pathnames for each of the attachments
    gm_Layout.GetSerializeFileName(t_strLayoutAttachment);
    gm_Connections.GetSerializeFileName(t_strConnectionsAttachment);
    gm_AppConfig.GetSerializeFileName(t_strConfigurationAttachment);
    GetFeedbackFileName(t_strFeedbackAttachment);

    // Generate the attachments list
    t_strAttachments.Format("%s;%s;%s;%s", t_strLayoutAttachment,
        t_strConnectionsAttachment, t_strConfigurationAttachment,
        t_strFeedbackAttachment);

    // Fill in subject and to and message
    t_strSubject.LoadString(IDS_DVSSSUPPORT);
    t_strMessage.LoadString(IDS_CONTACTINFO);
    t_strTo.LoadString(IDS_DVSSSUPPORTEMAIL);

    t_SimpleMAPI.QuickSendMail(t_strTo, "", t_strSubject,
        t_strMessage, t_strAttachments, true);

    return ;
}

// #import "C:\Program Files\Common
Files\InstallShield\UpdateService\Agent.exe" named_guids no_namespace
raw_interfaces_only
#include ".\release\Agent.tlh"
void CProjectNalayApp::OnHelpCheckForUpdates()
{
    USES_CONVERSION;
    CComPtr<IAgent2> spAgent;

```

```

        if (SUCCEEDED(spAgent.CoCreateInstance(CLSID_Agent)))
        {
            CComBSTR productCode(L"{E3D9A9CA-707B-48D6-8DAE-
525BFE0FFE5A}");
            HRESULT hr = spAgent->AppUpdate(productCode, AppMenu);
            if (FAILED(hr))
            {
                CComPtr<IErrorInfo> spErrorInfo;
                if (::GetErrorInfo(0, &spErrorInfo) == S_OK)
                {
                    CComBSTR bstrDescription;
                    if (SUCCEEDED(spErrorInfo->
>GetDescription(&bstrDescription)))
                    {
                        NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
"OnHelpCheckForUpdates", E_FAIL);
                        ::MessageBoxW(0, bstrDescription,
L"Error", 0);
                    }
                }
            }
            else
            {
                NSENDFEEDBACKHRESULT(FEEDBACKMESSAGETYPE_ERROR,
"OnHelpCheckForUpdates", E_FAIL);
            }
        }

// PropPageAudio.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageAudio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
////////
// CPropPageAudio property page

IMPLEMENT_DYNCREATE(CPropPageAudio, CPropertyPage)

CPropPageAudio::CPropPageAudio() : CPropertyPage(CPropPageAudio::IDD)
{
    //{AFX_DATA_INIT(CPropPageAudio)
    m_DefaultDirectory = _T("");
    //{AFX_DATA_INIT
}

CPropPageAudio::~CPropPageAudio()
{
}

```

```

void CPropPageAudio::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPropPageAudio)
    DDX_Text(pDX, IDC_DEFAULTDIRECTORY, m_DefaultDirectory);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageAudio, CPropertyPage)
    //{AFX_MSG_MAP(CPropPageAudio)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageAudio message handlers

BOOL CPropPageAudio::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    UpdateData(FALSE);
    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CPropPageAudio::OnBrowse()
{
    char t_szPath[_MAX_PATH];

    UpdateData(TRUE);

    lstrcpy(t_szPath, (LPCTSTR) m_DefaultDirectory);
    if ( PromptForFolder(m_hWnd, "Select Default Video Playback
Path", t_szPath) )
    {
        m_DefaultDirectory = t_szPath;
        UpdateData(FALSE);
    }
}

void CPropPageAudio::OnOK()
{
    UpdateData(TRUE);

    CPropertyPage::OnOK();
}
// PropPageConnections.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"

```

```

#include "PropPageConnections.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageConnections property page

IMPLEMENT_DYNCREATE(CPropPageConnections, CPropertyPage)

CPropPageConnections::CPropPageConnections() :
CPropertyPage(CPropPageConnections::IDD)

{
    //{AFX_DATA_INIT(CPropPageConnections)
    m_OnDoubleClick = -1;
    //}AFX_DATA_INIT
}

CPropPageConnections::~CPropPageConnections()
{
}

void CPropPageConnections::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPropPageConnections)
    DDX_Control(pDX, IDC_ONDOUBLECLICK, m_OnDoubleClickCtrl);
    DDX_CBIndex(pDX, IDC_ONDOUBLECLICK, m_OnDoubleClick);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageConnections, CPropertyPage)
    //{AFX_MSG_MAP(CPropPageConnections)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageConnections message handlers

BOOL CPropPageConnections::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // TODO: Add extra initialization here

    CString t_str;
    t_str.LoadString(IDS_OPENCONNECTION);
    m_OnDoubleClickCtrl.AddString(t_str);
    t_str.LoadString(IDS_CONFIGURECONNECTION);
    m_OnDoubleClickCtrl.AddString(t_str);
}

```

```

        UpdateData(FALSE);

        return TRUE; // return TRUE unless you set the focus to a
        control
                // EXCEPTION: OCX Property Pages should return
FALSE
    }

void CPropPageConnections::OnOK()
{
    UpdateData(TRUE);

    CPropertyPage::OnOK();
}
// PropPageEmail.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageEmail.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
/////
// CPropPageEmail property page

IMPLEMENT_DYNCREATE(CPropPageEmail, CPropertyPage)

CPropPageEmail::CPropPageEmail() : CPropertyPage(CPropPageEmail::IDD)
{
    //{{AFX_DATA_INIT(CPropPageEmail)
        // NOTE: the ClassWizard will add member initialization
here
    //}}AFX_DATA_INIT
}

CPropPageEmail::~CPropPageEmail()
{
}

void CPropPageEmail::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPropPageEmail)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageEmail, CPropertyPage)
    //{{AFX_MSG_MAP(CPropPageEmail)

```

```

        // NOTE: the ClassWizard will add message map macros here
    ///}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageEmail message handlers
// PropPageFeedback.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageFeedback.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageFeedback property page

IMPLEMENT_DYNCREATE(CPropPageFeedback, CPropertyPage)

CPropPageFeedback::CPropPageFeedback() :
CPropertyPage(CPropPageFeedback::IDD)
{
    //{{AFX_DATA_INIT(CPropPageFeedback)
    m_ShowDebugFeedback = FALSE;
    m_ShowErrorFeedback = FALSE;
    m_ShowStatusFeedback = FALSE;
    m_ShowWarningFeedback = FALSE;
    //}}AFX_DATA_INIT
}

CPropPageFeedback::~CPropPageFeedback()
{
}

void CPropPageFeedback::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPropPageFeedback)
    DDX_Check(pDX, IDC_SHOWDEBUGFEEDBACK, m_ShowDebugFeedback);
    DDX_Check(pDX, IDC_SHOWERRORFEEDBACK, m_ShowErrorFeedback);
    DDX_Check(pDX, IDC_SHOWSTATUSFEEDBACK, m_ShowStatusFeedback);
    DDX_Check(pDX, IDC_SHOWWARNINGFEEDBACK, m_ShowWarningFeedback);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageFeedback, CPropertyPage)
    //{{AFX_MSG_MAP(CPropPageFeedback)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP

```

```

END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageFeedback message handlers
// PropPagePhoneModem.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPagePhoneModem.h"

#include "TAPIControl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPagePhoneModem property page

IMPLEMENT_DYNCREATE(CPropPagePhoneModem, CPropertyPage)

CPropPagePhoneModem::CPropPagePhoneModem() :
CPropertyPage(CPropPagePhoneModem::IDD)
{
    ///{{AFX_DATA_INIT(CPropPagePhoneModem)
    ///}}AFX_DATA_INIT
}

CPropPagePhoneModem::~CPropPagePhoneModem()
{
}

void CPropPagePhoneModem::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CPropPagePhoneModem)
    DDX_Control(pDX, IDC_DEVICE, m_DeviceCtrl);
    ///}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPagePhoneModem, CPropertyPage)
    ///{{AFX_MSG_MAP(CPropPagePhoneModem)
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPagePhoneModem message handlers

void CPropPagePhoneModem::OnOK()
{

```

```

        int t_nCurSel;

        t_nCurSel = m_DeviceCtrl.GetCurSel();
        if ( t_nCurSel != CB_ERR )
            m_DeviceCtrl.GetLBText(t_nCurSel , m_TAPIDevice );

        CPropertyPage::OnOK();
    }

    BOOL CPropPagePhoneModem::OnInitDialog()
    {
        CPropertyPage::OnInitDialog();
        CTAPIControl t_TAPIControl;

        CStringArray t_arrDevices;

        t_TAPIControl.QuickGetLineNames(t_arrDevices);

        for (int i = 0; i < t_arrDevices.GetSize(); i++ )
            m_DeviceCtrl.AddString(t_arrDevices.ElementAt(i));

        m_DeviceCtrl.SelectString(-1, m_TAPIDevice);

        return TRUE; // return TRUE unless you set the focus to a
        control
        // EXCEPTION: OCX Property Pages should return
        FALSE
    }
    // PropPageRecordVideo.cpp : implementation file
    //

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageRecordVideo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageRecordVideo property page

IMPLEMENT_DYNCREATE(CPropPageRecordVideo, CPropertyPage)

CPropPageRecordVideo::CPropPageRecordVideo() :
CPropertyPage(CPropPageRecordVideo::IDD)

{
    //{AFX_DATA_INIT(CPropPageRecordVideo)
    m_DefaultRecordDuration = 0;
    m_DefaultRecordFile = _T("");
    m_MaxContinuousFiles = 0;
    //}}AFX_DATA_INIT

```

```

}

CPropPageRecordVideo::~CPropPageRecordVideo()
{
}

void CPropPageRecordVideo::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    ///{AFX_DATA_MAP(CPropPageRecordVideo)
    DDX_Control(pDX, IDC_DURATION, m_DefaultRecordDurationCtrl);
    DDX_DateTimeCtrl(pDX, IDC_DURATION, m_DefaultRecordDuration);
    DDX_Text(pDX, IDC_FILENAME, m_DefaultRecordFile);
    DDX_Text(pDX, IDC_MAXCONTINUOUSFILES, m_MaxContinuousFiles);
    DDV_MinMaxInt(pDX, m_MaxContinuousFiles, 1, 100);
    ///}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageRecordVideo, CPropertyPage)
    ///{AFX_MSG_MAP(CPropPageRecordVideo)
    ///}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageRecordVideo message handlers

void CPropPageRecordVideo::OnOK()
{
    UpdateData();

    CPropertyPage::OnOK();
}

BOOL CPropPageRecordVideo::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    CString t_strDateTimeFormat("HH':'mm':'ss");
    m_DefaultRecordDurationCtrl.SetFormat(t_strDateTimeFormat);

    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}
// PropPageSecurityPrompts.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageSecurityPrompts.h"

#ifdef _DEBUG

```

```

#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageSecurityPrompts property page

IMPLEMENT_DYNCREATE(CPropPageSecurityPrompts, CPropertyPage)

CPropPageSecurityPrompts::CPropPageSecurityPrompts() :
CPropertyPage(CPropPageSecurityPrompts::IDD)
{
    //{AFX_DATA_INIT(CPropPageSecurityPrompts)
    m_PwdPromptOnAlarm = FALSE;
    m_PwdPromptOnConfiguration = FALSE;
    m_PwdPromptOnConnection = FALSE;
    m_PwdPromptOnExit = FALSE;
    m_PwdPromptOnLaunch = FALSE;
    m_AppPassword = _T("");
    m_AppPassword2 = _T("");
    m_PromptOnExit = FALSE;
    m_EnableAppSecurityPrompt = FALSE;
    //}}AFX_DATA_INIT
}

CPropPageSecurityPrompts::~CPropPageSecurityPrompts()
{
}

void CPropPageSecurityPrompts::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPropPageSecurityPrompts)
    DDX_Control(pDX, IDC_PASSWORD2, m_Password2Ctrl);
    DDX_Control(pDX, IDC_PASSWORD, m_PasswordCtrl);
    DDX_Control(pDX, IDC_LAUNCH, m_LaunchCtrl);
    DDX_Control(pDX, IDC_EXIT, m_ExitCtrl);
    DDX_Control(pDX, IDC_CONNECTION, m_ConnectionCtrl);
    DDX_Control(pDX, IDC_CONFIGURATION, m_ConfigurationCtrl);
    DDX_Control(pDX, IDC_ALARM, m_AlarmCtrl);
    DDX_Check(pDX, IDC_ALARM, m_PwdPromptOnAlarm);
    DDX_Check(pDX, IDC_CONFIGURATION, m_PwdPromptOnConfiguration);
    DDX_Check(pDX, IDC_CONNECTION, m_PwdPromptOnConnection);
    DDX_Check(pDX, IDC_EXIT, m_PwdPromptOnExit);
    DDX_Check(pDX, IDC_LAUNCH, m_PwdPromptOnLaunch);
    DDX_Text(pDX, IDC_PASSWORD, m_AppPassword);
    DDX_Text(pDX, IDC_PASSWORD2, m_AppPassword2);
    DDX_Check(pDX, IDC_PROMPTONEXIT, m_PromptOnExit);
    DDX_Check(pDX, IDC_ENABLEAPPLICATIONSECURITYPROMPT,
m_EnableAppSecurityPrompt);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageSecurityPrompts, CPropertyPage)

```

```

//{{AFX_MSG_MAP(CPropPageSecurityPrompts)
    ON_BN_CLICKED(IDC_ENABLEAPPLICATIONSECURITYPROMPT,
OnEnableapplicationsecurityprompt)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageSecurityPrompts message handlers

void CPropPageSecurityPrompts::OnOK()
{
    CPropertyPage::OnOK();
}

BOOL CPropPageSecurityPrompts::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // TODO: Add extra initialization here

    UpdateCheckBoxes();

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCK Property Pages should return
FALSE
}

void CPropPageSecurityPrompts::OnEnableapplicationsecurityprompt()
{
    UpdateCheckBoxes();
}

void CPropPageSecurityPrompts::UpdateCheckBoxes()
{
    UpdateData(TRUE);

    if ( m_EnableAppSecurityPrompt )
    {
        m_Password2Ctrl.EnableWindow(TRUE);
        m_PasswordCtrl.EnableWindow(TRUE);
        m_LaunchCtrl.EnableWindow(TRUE);
        m_ExitCtrl.EnableWindow(TRUE);

        m_ConnectionCtrl.EnableWindow(TRUE);
        m_ConfigurationCtrl.EnableWindow(TRUE);
        m_AlarmCtrl.EnableWindow(TRUE);
    }
    else
    {
        m_Password2Ctrl.EnableWindow(FALSE);
        m_PasswordCtrl.EnableWindow(FALSE);
        m_LaunchCtrl.EnableWindow(FALSE);
        m_ExitCtrl.EnableWindow(FALSE);
        m_ConnectionCtrl.EnableWindow(FALSE);
        m_ConfigurationCtrl.EnableWindow(FALSE);
    }
}

```

```

        m_AlarmCtrl.EnableWindow(FALSE);
    }
}

BOOL CPropPageSecurityPrompts::OnApply()
{
    CString t_strMessage;

    UpdateData(TRUE);

    // If a password is selected, make sure it is confirmed and 4
digits
    if ( !m_AppPassword.IsEmpty() )
    {
        if ( m_AppPassword.GetLength() < 4 )
        {
            t_strMessage.LoadString(IDS_ERR_PASSWORDLENGTH);
            MessageBox(t_strMessage);
            return FALSE;
        }

        if ( m_AppPassword != m_AppPassword2 )
        {
            t_strMessage.LoadString(IDS_ERR_PASSWORDMATCH);
            MessageBox(t_strMessage);
            return FALSE;
        }
    }

    return CPropertyPage::OnApply();
}

// PropPageUpdate.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageUpdate.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageUpdate property page

IMPLEMENT_DYNCREATE(CPropPageUpdate, CPropertyPage)

CPropPageUpdate::CPropPageUpdate() :
CPropertyPage(CPropPageUpdate::IDD)
{
    //{AFX_DATA_INIT(CPropPageUpdate)
    m_CheckForUpdateAtLaunch = FALSE;
    //}AFX_DATA_INIT

```

```

}

CPropPageUpdate::~CPropPageUpdate()
{
}

void CPropPageUpdate::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPropPageUpdate)
    DDX_Check(pDX, IDC_CHECKUPDATEATLAUNCH,
m_CheckForUpdateAtLaunch);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageUpdate, CPropertyPage)
    //{AFX_MSG_MAP(CPropPageUpdate)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageUpdate message handlers
// PropPageVideo.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageVideo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageVideo property page

IMPLEMENT_DYNCREATE(CPropPageVideo, CPropertyPage)

CPropPageVideo::CPropPageVideo() : CPropertyPage(CPropPageVideo::IDD)
{
    //{AFX_DATA_INIT(CPropPageVideo)
    m_Format = -1;
    //}}AFX_DATA_INIT
}

CPropPageVideo::~CPropPageVideo()
{
}

void CPropPageVideo::DoDataExchange(CDataExchange* pDX)
{

```

```

        CPropertyPage::DoDataExchange(pDX);
        //{AFX_DATA_MAP(CPropPageVideo)
        DDX_CBIndex(pDX, IDC_FORMAT, m_Format);
        //}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CPropPageVideo, CPropertyPage)
    //{AFX_MSG_MAP(CPropPageVideo)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CPropPageVideo message handlers

void CPropPageVideo::OnOK()
{
    char t_szSubKey[] = "SOFTWARE\\CA505A\\CA505ACONTROL";
    char t_szValue[] = "VideoSetting";
    DWORD t_dwType = REG_DWORD;
    DWORD t_dwData;
    DWORD t_ldwData = sizeof(DWORD);

    UpdateData(TRUE);
    m_Format++;

    t_dwData = (DWORD) m_Format;
    SHSetValue(HKEY_LOCAL_MACHINE, (LPCTSTR) &t_szSubKey, (LPCTSTR)
&t_szValue, t_dwType, (LPVOID) &t_dwData, t_ldwData );

    CPropertyPage::OnOK();
}

BOOL CPropPageVideo::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    char t_szSubKey[] = "SOFTWARE\\CA505A\\CA505ACONTROL";
    char t_szValue[] = "VideoSetting";
    DWORD t_dwType = REG_DWORD;
    DWORD t_dwData;
    DWORD t_ldwData = sizeof(DWORD);
    DWORD t_dwReturn;

    t_dwReturn = SHGetValue(HKEY_LOCAL_MACHINE, (LPCTSTR)
&t_szSubKey, (LPCTSTR) &t_szValue, &t_dwType, (LPVOID) &t_dwData,
&t_ldwData );
    if ( t_dwReturn == ERROR_SUCCESS )
    {
        m_Format = (int) t_dwData - 1;
        UpdateData(FALSE);
    }

    return TRUE; // return TRUE unless you set the focus to a
control

```

```

// EXCEPTION: OCX Property Pages should return
FALSE
}
// PropPageVideoPlayback.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageVideoPlayback.h"

#include "SimpleVideo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CPropPageVideoPlayback property page

IMPLEMENT_DYNCREATE(CPropPageVideoPlayback, CPropertyPage)

CPropPageVideoPlayback::CPropPageVideoPlayback() :
CPropertyPage(CPropPageVideoPlayback::IDD)
{
    //{AFX_DATA_INIT(CPropPageVideoPlayback)
    m_FastForwardSpeed = 0;
    m_ForwardSpeed = 0;
    m_DefaultDirectory = _T("");
    //}}AFX_DATA_INIT
}

CPropPageVideoPlayback::~CPropPageVideoPlayback()
{
}

void CPropPageVideoPlayback::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPropPageVideoPlayback)
    DDX_Text(pDX, IDC_FASTFORWARDSPEED, m_FastForwardSpeed);
    DDV_MinMaxInt(pDX, m_FastForwardSpeed, 1, 10);
    DDX_Text(pDX, IDC_FORWARDSPEED, m_ForwardSpeed);
    DDV_MinMaxInt(pDX, m_ForwardSpeed, 1, 5);
    DDX_Text(pDX, IDC_DEFAULTDIRECTORY, m_DefaultDirectory);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropPageVideoPlayback, CPropertyPage)
    //{AFX_MSG_MAP(CPropPageVideoPlayback)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageVideoPlayback message handlers

void CPropPageVideoPlayback::OnOK()
{
    UpdateData(TRUE);

    CPropertyPage::OnOK();
}

BOOL CPropPageVideoPlayback::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    UpdateData(FALSE);
    return TRUE; // return TRUE unless you set the focus to a
control
// EXCEPTION: OCX Property Pages should return
FALSE
}

void CPropPageVideoPlayback::OnBrowse()
{
    char t_szPath[_MAX_PATH];

    UpdateData(TRUE);

    lstrcpy(t_szPath, (LPCTSTR) m_DefaultDirectory);
    if ( PromptForFolder(m_hWnd, "Select Default Video Playback
Path", t_szPath) )
    {
        m_DefaultDirectory = t_szPath;
        UpdateData(FALSE);
    }
}

// PropPageYellowPages.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "PropPageYellowPages.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
// CPropPageYellowPages property page

IMPLEMENT_DYNCREATE(CPropPageYellowPages, CPropertyPage)

```

[illegible]

```
IMPLEMENT_DYNCREATE(CRemoteVideoEventsView, CListView)
```

```
CRemoteVideoEventsView::CRemoteVideoEventsView()
```

```
{
}
```

```
CRemoteVideoEventsView::~CRemoteVideoEventsView()
```

```
{
}
```

```
BEGIN_MESSAGE_MAP(CRemoteVideoEventsView, CListView)
```

```
    ///{{AFX_MSG_MAP(CRemoteVideoEventsView)
```

```
    ON_COMMAND(ID_CONNECTION_CONNECT, OnConnectionConnect)
```

```
    ON_COMMAND(ID_CONNECTION_DISCONNECT, OnConnectionDisconnect)
```

```
    ON_COMMAND(ID_CONNECTION_DELETE_EVENT, OnConnectionDeleteEvent)
```

```
    ON_COMMAND(ID_CONNECTION_NEW_ALARM, OnConnectionNewAlarm)
```

```
    ON_COMMAND(ID_CONNECTION_NEW_EVENT, OnConnectionNewEvent)
```

```
    ON_COMMAND(ID_VIEW_LARGEICONS, OnViewLargeicons)
```

```
    ON_COMMAND(ID_VIEW_SMALLICONS, OnViewSmallicons)
```

```
    ON_COMMAND(ID_VIEW_LIST, OnViewList)
```

```
    ON_COMMAND(ID_VIEW_DETAILS, OnViewDetails)
```

```
    ON_NOTIFY_REFLECT(NM_DBLCLK, OnDbclclk)
```

```
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_CONNECT,
```

```
OnUpdateConnectionConnect)
```

```
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_DISCONNECT,
```

```
OnUpdateConnectionDisconnect)
```

```
    ON_WM_DESTROY()
```

```
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_NEW_ALARM,
```

```
OnUpdateConnectionNewAlarm)
```

```
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_NEW_EVENT,
```

```
OnUpdateConnectionNewEvent)
```

```
    ON_UPDATE_COMMAND_UI(ID_CONNECTION_DELETE_EVENT,
```

```
OnUpdateConnectionDeleteEvent)
```

```
    ///}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
/////
```

```
// CRemoteVideoEventsView drawing
```

```
void CRemoteVideoEventsView::OnDraw(CDC* pDC)
```

```
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}
```

```
////////////////////////////////////
/////
```

```
// CRemoteVideoEventsView diagnostics
```

```
#ifdef _DEBUG
```

```
void CRemoteVideoEventsView::AssertValid() const
```

```
{
    CListView::AssertValid();
}
```

```

void CRemoteVideoEventsView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}

#endif //_DEBUG

////////////////////////////////////
////////
// CRemoteVideoEventsView message handlers

void CRemoteVideoEventsView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    CListCtrl& t_ctlList = GetListCtrl();
    CString t_strItem;

    m_LargeImageList.Create(IDB_EVENTSLARGEICONS, 32, 1, RGB(255,
255, 255));
    m_SmallImageList.Create(IDB_EVENTSSMALLICONS, 16, 1, RGB(255,
255, 255));
    // m_StateImageList.Create(IDB_CONNECTIONSSTATEICONS, 8, 1, RGB(255,
0, 0));

    t_ctlList.SetImageList(&m_LargeImageList, LVSIL_NORMAL);
    t_ctlList.SetImageList(&m_SmallImageList, LVSIL_SMALL);
    // t_ctlList.SetImageList(&m_StateImageList, LVSIL_STATE);

    // Setup columns
    t_strItem.LoadString(IDS_LABEL);
    t_ctlList.InsertColumn(0, t_strItem);
    SetColumnWidth(0, 110);
    t_strItem.LoadString(IDS_TYPE);
    t_ctlList.InsertColumn(1, t_strItem);
    SetColumnWidth(1, 110);
    t_strItem.LoadString(IDS_DETAILS);
    t_ctlList.InsertColumn(2, t_strItem);
    SetColumnWidth(2, 450);
}

BOOL CRemoteVideoEventsView::PreCreateWindow(CREATESTRUCT& cs)
{
    // Default to report view
    cs.style |= LVS_ICON | LVS_NOSORTHEADER | LVS_SINGLESEL;

    return CListView::PreCreateWindow(cs);
}

void CRemoteVideoEventsView::OnConnectionConnect()
{
    // Connect to video first
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUUPDATE_CONNECT, NULL);
}

```

```

        // Connect DirectPlay
        CString t_strConnectionLab l;
        if ( !GetConnectionLabel(t_strConnectionLabel) )
            return;

        if ( !gm_Connections.Connect(t_strConnectionLabel, m_hWnd) )
            t_pDoc->UpdateAllViews(NULL, NUPDATE_DISCONNECT, NULL);
    }

void CRemoteVideoEventsView::OnConnectionDisconnect()
{
    // Disconnect to video first
    CDocument* t_pDoc = GetDocument();
    if ( t_pDoc != NULL )
        t_pDoc->UpdateAllViews(NULL, NUPDATE_DISCONNECT, NULL);

    // Disconnect DirectPlay
    CString t_strConnectionLabel;
    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;
    gm_Connections.Disconnect(t_strConnectionLabel);

    // Clear the Events
    m_Events.RemoveAll();
    UpdateListCtrl();
}

void CRemoteVideoEventsView::OnConnectionDeleteEvent()
{
    CString t_strEventLabel;

    // Get label of selected item and delete it
    if ( GetSelectedEventLabel(t_strEventLabel) )
        if ( m_Events.DeleteEvent(t_strEventLabel) )
        {
            UpdateListCtrl();
            SendEventsArray();
        }
}

void CRemoteVideoEventsView::OnConnectionNewAlarm()
{
    CEventDialog t_EventDialog;
    CEventInfo t_EventInfo;
    CString t_strConnectionLabel;

    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;

    // Connection must be established - no use to create alarms on a
    remote connection
    // unless it is transmitted
    if (
!gm_Connections.IsConnectionEstablished(t_strConnectionLabel) )
        return;

    t_EventDialog.m_EventInfo.m_EventType = EVENTTYPE_ALARM;

```

```

    t_EventDialog.m_ConnectionLabel = t_strConnectionLabel;

    if ( t_EventDialog.DoModal() == IDOK )
    {
        t_EventInfo = t_EventDialog.m_EventInfo;
        m_Events.Add(t_EventInfo);
        UpdateListCtrl();
        SendEventsArray();
    }

    return;
}

void CRemoteVideoEventsView::OnConnectionNewEvent()
{
    bool t_bReturn = false;
    CEventDialog t_EventDialog;
    CEventInfo t_EventInfo;
    CString t_strConnectionLabel;

    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;

    // Connection must be established - no use to create alarms on a
remote connection
    // unless it is transmitted
    if (
!gm_Connections.IsConnectionEstablished(t_strConnectionLabel) )
        return;

    t_EventDialog.m_EventInfo.m_EventType = EVENTTYPE_SCHEDULEDEVENT;
    t_EventDialog.m_ConnectionLabel = t_strConnectionLabel;

    if ( t_EventDialog.DoModal() == IDOK )
    {
        t_EventInfo = t_EventDialog.m_EventInfo;
        m_Events.Add(t_EventInfo);
        UpdateListCtrl();
        SendEventsArray();
    }

    return ;
}

bool CRemoteVideoEventsView::GetConnectionLabel(CString&
strConnectionLabel)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    bool t_bReturn = false;

    if ( t_LocalVideoDoc == NULL )
        goto Exit1;

    strConnectionLabel = t_LocalVideoDoc->m_ConnectionLabel;

    t_bReturn = true;
}

```

```

Exit1:
    return t_bReturn;
}

void CRemoteVideoEventsView::OnViewLargeicons()
{
    if (GetViewType() != LVS_ICON)
        SetViewType(LVS_ICON);
}

void CRemoteVideoEventsView::OnViewSmallicons()
{
    if (GetViewType() != LVS_SMALLICON)
        SetViewType(LVS_SMALLICON);
}

void CRemoteVideoEventsView::OnViewList()
{
    if (GetViewType() != LVS_LIST)
        SetViewType(LVS_LIST);
}

void CRemoteVideoEventsView::OnViewDetails()
{
    if (GetViewType() != LVS_REPORT)
        SetViewType(LVS_REPORT);
}

BOOL CRemoteVideoEventsView::SetViewType(DWORD dwViewType)
{
    return(ModifyStyle(LVS_TYPEMASK, dwViewType & LVS_TYPEMASK));
}

DWORD CRemoteVideoEventsView::GetViewType()
{
    return(GetStyle() & LVS_TYPEMASK);
}

BOOL CRemoteVideoEventsView::SetColumnWidth(int Column, int Width)
{
    CListCtrl& t_ctlList = GetListCtrl();
    LV_COLUMN t_lvColumn;

    t_lvColumn.mask = LVCF_WIDTH;
    t_lvColumn.cx = Width;
    return t_ctlList.SetColumn(Column, &t_lvColumn);
}

bool CRemoteVideoEventsView::UpdateListCtrl()
{
    bool t_bReturn = false;
    int i;
    CString t_str;
    CListCtrl& t_ctlList = GetListCtrl();

    // Clear contents of control

```

```

t_ctlList.DeleteAllItems();

// Get array of connections
for ( i = 0; i < m_Events.GetSize(); i++ )
{
    CEventInfo& t_EventInfo = m_Events[i];

    // Insert item
    t_ctlList.InsertItem( LVIF_TEXT | LVIF_IMAGE ,
                        i, t_EventInfo.m_Label,
                        NULL, NULL,
t_EventInfo.m_EventType,
                        NULL);

    // Set the remaining fields
    t_EventInfo.GetEventTypeString(t_str);
    t_ctlList.SetItemText(i, 1, t_str);
    t_EventInfo.GetEventDetailString(t_str);
    t_ctlList.SetItemText(i, 2, t_str);
}

t_bReturn = true;
//Exit1:
return t_bReturn;
}

void CRemoteVideoEventsView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    switch ( lHint )
    {
    case NUPDATE_REFRESH:
        UpdateListCtrl();
        break;
    case NUPDATE_VIEWDETAILS:
        OnViewDetails();
        break;
    case NUPDATE_VIEWLARGEICONS:
        OnViewLargeicons() ;
        break;
    case NUPDATE_VIEWLIST:
        OnViewList() ;
        break;
    case NUPDATE_VIEWSMALLICONS:
        OnViewSmallicons();
        break;
    case NUPDATE_NEWALARM:
        OnConnectionNewAlarm();
        break;
    case NUPDATE_NEWSCHEDULEDEVENT:
        OnConnectionNewEvent();
        break;
    case NUPDATE_DELETEEVENT:
        OnConnectionDeleteEvent();
        break;
    // case NUPDATE_IMMESSAGERECEIVED:
    case DPN_MSGID_RECEIVE:

```

```

        {
            CConnectionMsg* t_pConn ctionMsg;
            t_pConnectionMsg = (CConnectionMsg*) pHint;
            ReceiveEventsArray(t_pConnectionMsg);
        }
        break;
    };
}

void CRemoteVideoEventsView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    OnEditEvent();

    *pResult = 0;
}

void CRemoteVideoEventsView::OnEditEvent()
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    CString t_strEventLabel;

    if ( t_LocalVideoDoc == NULL )
        return;

    // Get label of selected item and delete it
    if ( GetSelectedEventLabel(t_strEventLabel) )
        if ( m_Events.EditEvent(t_LocalVideoDoc->m_ConnectionLabel,
t_strEventLabel) )
        {
            UpdateListCtrl();
            SendEventsArray();
        }
}

bool CRemoteVideoEventsView::GetSelectedEventLabel(CString& strLabel)
{
    bool t_bReturn = false;

    CListCtrl& t_ctlList = GetListCtrl();
    int t_nItem ;

    // Get selected item - this is a single-selection list control
    t_nItem = GetSelectedEventItem();
    if ( t_nItem == -1 )
        goto Exit1;

    // Get the m_Connections array index for that item
    strLabel = t_ctlList.GetItemText(t_nItem, 0);

    t_bReturn = true;
Exit1:
    return t_bReturn;
}

int CRemoteVideoEventsView::GetSelectedEventItem()
{

```

```

CListCtrl& t_ctlList = GetListCtrl();
int t_nItem = -1;

// Get selected item - this is a single-selection list control
POSITION t_pos = t_ctlList.GetFirstSelectedItemPosition();

// Validate that an item was selected
if (t_pos == NULL)
{
    NSENDFEEDBACKIDS(FEEDBACKMESSAGETYPE_WARNING, "",
IDS_NOCONNECTIONSELECTED);
    goto Exit1;
}

// Get the item number selected
t_nItem = t_ctlList.GetNextSelectedItem(t_pos);

Exit1:
    return t_nItem;
}

bool CRemoteVideoEventsView::ReceiveEventsArray(CConnectionMsg*
pConnectionMsg)
{
    bool t_bReturn = false;
    CMemFile t_MemFile;
    CListCtrl& t_ctlList = GetListCtrl();
    BYTE t_pData[CONNECTIONMSG_BUFFERLENGTH];

    memcpy(t_pData, pConnectionMsg->m_pData,
CONNECTIONMSG_BUFFERLENGTH);
    t_MemFile.Attach(&t_pData[0], CONNECTIONMSG_BUFFERLENGTH);
    CArchive t_archive(&t_MemFile, CArchive::load);

    t_archive.Flush();
    t_MemFile.Flush();

    m_Events.RemoveAll();
    m_Events.Serialize( t_archive);

    t_archive.Flush();
    t_MemFile.Flush();

    t_MemFile.Detach();

    t_MemFile.Close();

    UpdateListCtrl();

    t_bReturn = true;
//Exit1:

    return t_bReturn;
}

bool CRemoteVideoEventsView::SendEventsArray()
{

```

```

bool t_bReturn = false;
CString t_strConn ctionLabel;
CConnectionMsg t_ConnectionMsg;
CString t_strDefaultFilename;
CMemFile t_MemFile;
DWORD t_dwMemSize;
BYTE* t_pData;

if ( !GetConnectionLabel(t_strConnectionLabel) )
    return false;

CArchive t_archive(&t_MemFile, CArchive::store);
m_Events.Serialize(t_archive );

t_archive.Flush();
t_MemFile.Flush();
t_dwMemSize = t_MemFile.GetLength();

t_pData = t_MemFile.Detach();
memcpy(t_ConnectionMsg.m_pData, t_pData, t_dwMemSize);
free(t_pData);

// Setup message handler
t_ConnectionMsg.m_nMessageType =
CONNECTIONMSGTYPE_LOCALVIDEOEVENTS;
t_ConnectionMsg.SetLabel(t_strConnectionLabel);
t_ConnectionMsg.m_bLocal = true;

// Send message to other connections
gm_Connections.SendMessage(t_strConnectionLabel,
t_ConnectionMsg);

t_bReturn = true;
//Exit1:
return t_bReturn;
}

void CRemoteVideoEventsView::OnUpdateConnectionConnect(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
!gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CRemoteVideoEventsView::OnUpdateConnectionDisconnect(CCmdUI*
pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )

```

```

        pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
    }

void CRemoteVideoEventsView::OnDestroy()
{
    CListView::OnDestroy();

    CString t_strConnectionLabel;
    if ( !GetConnectionLabel(t_strConnectionLabel) )
        return;
    OnConnectionDisconnect();
}

void CRemoteVideoEventsView::OnUpdateConnectionNewAlarm(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CRemoteVideoEventsView::OnUpdateConnectionNewEvent(CCmdUI* pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}

void CRemoteVideoEventsView::OnUpdateConnectionDeleteEvent(CCmdUI*
pCmdUI)
{
    CLocalVideoDoc * t_LocalVideoDoc = (CLocalVideoDoc * )
GetDocument();
    if ( t_LocalVideoDoc == NULL ) return;
    if ( !t_LocalVideoDoc->m_ConnectionLabel.IsEmpty() )
        pCmdUI->Enable(
gm_Connections.IsConnectionEstablished(t_LocalVideoDoc-
>m_ConnectionLabel) );
}
// RemoteVideoFrame.cpp : implementation file
//

#include "stdafx.h"
#include "Project Nalay.h"
#include "RemoteVideoFrame.h"
#include "RemoteVideoWMPView.h"

#ifdef _DEBUG

```

```

#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CRemoteVideoFrame

IMPLEMENT_DYNCREATE(CRemoteVideoFrame, CMDIChildWnd)

CRemoteVideoFrame::CRemoteVideoFrame()
{
}

CRemoteVideoFrame::~CRemoteVideoFrame()
{
}

BEGIN_MESSAGE_MAP(CRemoteVideoFrame, CMDIChildWnd)
    //{AFX_MSG_MAP(CRemoteVideoFrame)
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CRemoteVideoFrame message handlers

BOOL CRemoteVideoFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // create a splitter with 2 rows, 1 column.
    if (!m_wndSplitter.CreateStatic(this, 1, 2))
    {
        TRACE0("Failed to CreateStaticSplitter\n");
        return FALSE;
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
        pContext->m_pNewViewClass, CSize(130, 50), pContext))
    {
        TRACE0("Failed to create first pane\n");
        return FALSE;
    }

    // add the second splitter pane - an input view in row 0
    if (!m_wndSplitter.CreateView(0, 1,
        RUNTIME_CLASS(CRemoteVideoWMPView), CSize(0, 0), pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }
}

```

```
// activate the input view
SetActiveView((CView*)m_wndSplitter.GetPane(0,0));

return TRUE;
}

BOOL CRemoteVideoFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    return CMDIChildWnd::PreCreateWindow(cs);
}

int CRemoteVideoFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this,
        CBRS_TOP|CBRS_TOOLTIPS|CBRS_FLYBY|WS_VISIBLE) ||
        !m_wndToolBar.LoadToolBar(IDR_REMOTEVIDEOSURVEILLANCE))
    {
        return FALSE;        // fail to create
    }

    return 0;
}
```

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Motion Detection Filter.rc
//
#define VERSION_RES_MINOR_VER          0
#define VERSION_RES_BUILD              0
#define VER_DEBUG                      0
#define VERSION_RES_MAJOR_VER         1
#define IDS_TITLE                      1
#define IDD_PROPPAGE_MAIN              104
#define IDC_NUMSEGMENTS                1000
#define IDC_INTERVALPERIOD             1001
#define IDC_GRANULARITY                1002
#define IDC_COLORERRORMARGIN           1003
#define IDC_PERCENTTHRESHOLDHIGH       1004
#define IDC_PERCENTTHRESHOLDLOW       1005
#define IDC_OPNONE                     1006
#define IDC_OPSTANDARD                 1007
#define IDC_OPFIXED                    1008
#define IDC_DWELLTIME                  1009
#define IDC_SENSITIVITY                1010
#define VERSION_RES_LANGUAGE           0x409
#define VERSION_RES_CHARSET            1252
#define IDC_STATIC                     -1

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC                    1
#define _APS_NEXT_RESOURCE_VALUE       102
#define _APS_NEXT_COMMAND_VALUE       40001
#define _APS_NEXT_CONTROL_VALUE       1011
#define _APS_NEXT_SYMED_VALUE         101
#endif
#endif
//-----
//
// Class: CPropPageMain
//
// Description: Main property page for controlling motion detection
// variables
//
//
// History: 02/12/02    LCK          Created
//           02/13/02    LCK          Added support for type of
// operation
//
//-----
//-----

#include <streams.h>

// Eliminate two expected level 4 warnings from the Microsoft compiler.
// The class does not have an assignment or copy operator, and so
// cannot

```

```

// be passed by value. This is normal. This file compiles clean at
the
// highest (most picky) warning level (-W4).
#pragma warning(disable: 4511 4512)

#include <windowsx.h>
#include <commctrl.h>
#include <olectl.h>
#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <tchar.h>

#include "resource.h" // ids used in the dialog
#include "Motion Detection Filter GUIDs.h" // public guids
#include "Motion Detection Filter Prop Page.h" // our own class

//
// CreateInstance
//
// Override CClassFactory method.
// Set lpUnk to point to an IUnknown interface on a new CPropPageMain
object
// Part of the COM object instantiation mechanism
//
CUnknown * WINAPI CPropPageMain::CreateInstance(LPUNKNOWN lpunk,
HRESULT *phr)
{
    CUnknown *punk = new CPropPageMain(lpunk, phr);
    if (punk == NULL) {
        *phr = E_OUTOFMEMORY;
    }
    return punk;
}

//
// CPropPageMain::Constructor
//
// Constructs and initialises a CPropPageMain object
//
CPropPageMain::CPropPageMain(LPUNKNOWN pUnk, HRESULT *phr)
    : CBasePropertyPage(NAME("NullIP Property Page"), pUnk,
        IDD_PROPPAGE_MAIN, IDS_TITLE)
{
    ASSERT(phr);
    m_pIIPMDFControlVariables = NULL;
    m_bIsInitialized = FALSE;
} // (constructor) CPropPageMain

//
// OnReceiveMessage
//
// Handles the messages for our property window

```

```

//
BOOL CPropPageMain::OnReceiveMessage(HWND hwnd, UINT uMsg, WPARAM
wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_COMMAND:
        {
            if (m_bIsInitialized)
            {
                m_bDirty = TRUE;
                if (m_pPageSite)
                {
                    m_pPageSite->OnStatusChange(PROPPAGESTATUS_DIRTY);
                }
            }
            return (LRESULT) TRUE;
        }
    }
    return
CBasePropertyPage::OnReceiveMessage(hwnd, uMsg, wParam, lParam);
} // OnReceiveMessage

//
// OnConnect
//
// Called when we connect to a transform filter
//
HRESULT CPropPageMain::OnConnect(IUnknown *pUnknown)
{
    ASSERT(m_pIIPMDFControlVariables == NULL);

    HRESULT hr = pUnknown->QueryInterface(IID_MDF_CONTROLVARIABLES,
(void **) &m_pIIPMDFControlVariables );
    if (FAILED(hr))
    {
        return E_NOINTERFACE;
    }

    ASSERT(m_pIIPMDFControlVariables);

    // Get the initial controlling values
    m_pIIPMDFControlVariables->get_NumSegments(&m_nNumSegments);
    m_pIIPMDFControlVariables-
>get_IntervalPeriod(&m_nIntervalPeriod);
    m_pIIPMDFControlVariables->get_Granularity(&m_nGranularity);
    m_pIIPMDFControlVariables-
>get_ColorErrorMargin(&m_nColorErrorMargin);
    m_pIIPMDFControlVariables-
>get_PercentThresholdHigh(&m_nPercentThresholdHigh);
    m_pIIPMDFControlVariables-
>get_PercentThresholdLow(&m_nPercentThresholdLow);
    m_pIIPMDFControlVariables->get_Operation(&m_nOperation);
    m_pIIPMDFControlVariables->get_DwellTime(&m_nDwellTime);

```

```

        m_bIsInitialized = FALSE ;
        return NOERROR;
    } // OnConnect

//
// OnDisconnect
//

// Likewise called when we disconnect from a filter
//
HRESULT CPropPageMain::OnDisconnect()
{
    if (m_pIIPMDFControlVariables == NULL) {
        return E_UNEXPECTED;
    }

    m_pIIPMDFControlVariables->Release();
    m_pIIPMDFControlVariables = NULL;

    return NOERROR;
} // OnDisconnect

//
// OnActivate
//
// We are being activated
//
HRESULT CPropPageMain::OnActivate()
{
    // Set controls with current values
    SetControlValues();

    m_bIsInitialized = TRUE;
    return NOERROR;
} // OnActivate

//
// OnDeactivate
//
// We are being deactivated
//
HRESULT CPropPageMain::OnDeactivate(void)
{
    m_bIsInitialized = FALSE;
    return NOERROR;
} // OnDeactivate

//
// OnApplyChanges
//
// Apply any changes so far made
// Also called when OnOK is called

```

```

//
HRESULT CPropPageMain::OnApplyChange s()
{
    TCHAR t_sz[STR_MAX_LENGTH];

    // Retrive Number of Segments, validate and update filter
    Edit_GetText(GetDlgItem(m_Dlg, IDC_NUMSEGMENTS), t_sz,
STR_MAX_LENGTH);
    m_nNumSegments = atoi(t_sz);
    if ( m_nNumSegments < MDF_MIN_NUMSEGMENTS ) m_nNumSegments =
MDF_MIN_NUMSEGMENTS ;
    if ( m_nNumSegments > MDF_MAX_NUMSEGMENTS ) m_nNumSegments =
MDF_MAX_NUMSEGMENTS ;
    m_pIIPMDFControlVariables->put_NumSegments(m_nNumSegments);

    // Retrive Interval Period, validate and update filter
    Edit_GetText(GetDlgItem(m_Dlg, IDC_INTERVALPERIOD), t_sz,
STR_MAX_LENGTH);
    m_nIntervalPeriod = atoi(t_sz);
    if ( m_nIntervalPeriod < MDF_MIN_INTERVALPERIOD )
m_nIntervalPeriod = MDF_MIN_INTERVALPERIOD ;
    if ( m_nIntervalPeriod > MDF_MAX_INTERVALPERIOD )
m_nIntervalPeriod = MDF_MAX_INTERVALPERIOD ;
    m_pIIPMDFControlVariables->put_IntervalPeriod(m_nIntervalPeriod );

    // Retrive Granularity, validate and update filter
    Edit_GetText(GetDlgItem(m_Dlg, IDC_GRANULARITY), t_sz,
STR_MAX_LENGTH);
    m_nGranularity = atoi(t_sz);
    if ( m_nGranularity < MDF_MIN_GRANULARITY ) m_nGranularity =
MDF_MIN GRANULARITY;
    if ( m_nGranularity > MDF_MAX_GRANULARITY ) m_nGranularity =
MDF_MAX GRANULARITY;
    m_pIIPMDFControlVariables->put_Granularity(m_nGranularity );

    // Retrive Color Error Margin, validate and update filter
    Edit_GetText(GetDlgItem(m_Dlg, IDC_COLORERRORMARGIN), t_sz,
STR_MAX_LENGTH);
    m_nColorErrorMargin = atoi(t_sz);
    if ( m_nColorErrorMargin < MDF_MIN_COLORERRORMARGIN )
m_nColorErrorMargin = MDF_MIN_COLORERRORMARGIN ;
    if ( m_nColorErrorMargin > MDF_MAX_COLORERRORMARGIN )
m_nColorErrorMargin = MDF_MAX_COLORERRORMARGIN;
    m_pIIPMDFControlVariables->put_ColorErrorMargin(m_nColorErrorMargin
);

    // Retrive Percent Threshold High, validate and update filter
    Edit_GetText(GetDlgItem(m_Dlg, IDC_PERCENTTHRESHOLDHIGH), t_sz,
STR_MAX_LENGTH);
    m_nPercentThresholdHigh = atoi(t_sz);
    if ( m_nPercentThresholdHigh < MDF_MIN_PERCENTTHRESHOLDHIGH )
m_nPercentThresholdHigh = MDF_MIN_PERCENTTHRESHOLDHIGH;
    if ( m_nPercentThresholdHigh > MDF_MAX_PERCENTTHRESHOLDHIGH )
m_nPercentThresholdHigh = MDF_MAX_PERCENTTHRESHOLDHIGH;
    m_pIIPMDFControlVariables-
>put_PercentThresholdHigh(m_nPercentThresholdHigh );

```

```

        // Retrive Percent Threshold Low, validate and update filter
        Edit_GetText(GetDlgItem(m_Dlg, IDC_PERCENTTHRESHOLDLOW), t_sz,
STR_MAX_LENGTH);
        m_nPercentThresholdLow = atoi(t_sz);
        if ( m_nPercentThresholdLow < MDF_MIN_PERCENTTHRESHOLDLOW)
m_nPercentThresholdLow = MDF_MIN_PERCENTTHRESHOLDLOW;
        if ( m_nPercentThresholdLow > MDF_MAX_PERCENTTHRESHOLDLOW)
m_nPercentThresholdLow = MDF_MAX_PERCENTTHRESHOLDLOW;
        m_pIIPMDFControlVariables->
>put_PercentThresholdLow(m_nPercentThresholdLow );

        // Find which operation we have selected and update filter
        for (int i = IDC_OPNONE; i <= IDC_OPFIXED; i++)
        {
            if (IsDlgButtonChecked(m_Dlg, i))
            {
                m_nOperation = i;
                break;
            }
        }
        m_pIIPMDFControlVariables->put_Operation(m_nOperation );

        // Retrive Dwell Time, validate and update filter
        Edit_GetText(GetDlgItem(m_Dlg, IDC_DWELLTIME), t_sz,
STR_MAX_LENGTH);
        m_nDwellTime = atoi(t_sz);
        if ( m_nDwellTime < MDF_MIN_DWELLTIME ) m_nDwellTime =
MDF_MIN_DWELLTIME;
        if ( m_nDwellTime > MDF_MAX_DWELLTIME ) m_nDwellTime =
MDF_MAX_DWELLTIME;
        m_pIIPMDFControlVariables->put_DwellTime(m_nDwellTime );

        // Set controls with current values in case they were corrected
        // by this function for min/max
        SetControlValues();

        return NOERROR;
    } // OnApplyChanges

//
// SetControlValues
//
// Updates the controls with the current variable values
//
HRESULT CPropPageMain::SetControlValues()
{
    TCHAR t_sz[STR_MAX_LENGTH];

    // Set the controls to the variable values

    // Set Number of Segments
    _stprintf(t_sz, TEXT("%d"), m_nNumSegments);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_NUMSEGMENTS), t_sz);

    // Set Interval Period
    _stprintf(t_sz, TEXT("%d"), m_nIntervalPeriod);

```

```

Edit_SetText(GetDlgItem(m_Dlg, IDC_INTERVALPERIOD), t_sz);

    // Set Granularity
    _stprintf(t_sz, TEXT("%d"), m_nGranularity);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_GRANULARITY), t_sz);

    // Set Color Error Margin
    _stprintf(t_sz, TEXT("%d"), m_nColorErrorMargin);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_COLORERRORMARGIN), t_sz);

    // Set Percent Threshold High
    _stprintf(t_sz, TEXT("%d"), m_nPercentThresholdHigh);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_PERCENTTHRESHOLDHIGH), t_sz);

    // Set Percent Threshold Low
    _stprintf(t_sz, TEXT("%d"), m_nPercentThresholdLow);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_PERCENTTHRESHOLDLOW), t_sz);

    // Set operation
    CheckRadioButton(m_Dlg, IDC_OPNONE, IDC_OPFIXED, m_nOperation);

    // Set Dwell Time
    _stprintf(t_sz, TEXT("%d"), m_nDwellTime);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_DWELLTIME), t_sz);

    return NOERROR;
} // SetControlValues
//-----
//
// Class: CPropPageMain
//
// Description: Main property page for controlling motion detection
// variables
//
//
// History: 02/12/02    LCK    Created
//           02/13/02    LCK    Added support for type of
// operation
//
//-----
//-----

#define MDF_MIN_NUMSEGMENTS          1
#define MDF_MAX_NUMSEGMENTS          10
#define MDF_MIN_INTERVALPERIOD        1
#define MDF_MAX_INTERVALPERIOD       1000
#define MDF_MIN_GRANULARITY           1
#define MDF_MAX_GRANULARITY          100
#define MDF_MIN_PERCENTTHRESHOLDLOW  1
#define MDF_MAX_PERCENTTHRESHOLDLOW  50
#define MDF_MIN_PERCENTTHRESHOLDHIGH 50
#define MDF_MAX_PERCENTTHRESHOLDHIGH 100
#define MDF_MIN_COLORERRORMARGIN      1
#define MDF_MAX_COLORERRORMARGIN      50
#define MDF_MIN_DWELLTIME              1
#define MDF_MAX_DWELLTIME             3600

```

```

class CPropPageMain : public CBase PropertyPage
{
public:

    static CUnknown * WINAPI CreateInstance(LPUNKNOWN lpunk, HRESULT
*phr);
    DECLARE_IUNKNOWN;

private:
    // Constructor
    CPropPageMain(LPUNKNOWN lpunk, HRESULT *phr);

    // CBasePropertyPage override functions
    BOOL OnReceiveMessage(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam);
    HRESULT OnConnect(IUnknown *pUnknown);
    HRESULT OnDisconnect();
    HRESULT OnActivate();
    HRESULT OnDeactivate();
    HRESULT OnApplyChanges();

    // Interval variables and functions
    HRESULT SetControlValues();

    BOOL m_bIsInitialized; // Used to ignore startup messages

    // Main Variables that are controlled by this prop page
    int m_nNumSegments; // The number of
segments that the video frame will be divided into
    int m_nIntervalPeriod; // Number of
seconds between checking for motion changes
    int m_nGranularity; // Density of
pixels checked for motion change
    int m_nPercentThresholdHigh; // High threshold %
change above which motion detection is invalid
    int m_nPercentThresholdLow; // Low threshold %
change above which motion detection is invalid
    int m_nColorErrorMargin; // Error is the amount
of color differential allowed when comparing green, blue or red
    int m_nOperation; // Which
operation user selected
    int m_nDwellTime; // Amount of time
between motion detection notifications

    IIPMDFControlVariables *m_pIIPMDFControlVariables;

}; // class NullIPProperties

//-----
// File: Motion Detection Filter.cpp
//
// Desc: DirectShow filter for detecting motion in a video stream
//

```

```

// Comments:      This filter is a basic DirectShow in-plac transfer
filter that takes a stream
//                and checks for pixel changes between frames to
det rmine whether there is motion.
//                While this method is simple to implement, it is
limited in the fact that there may be noise
//                that can easily be mistaken for motion.
Therefore, a set of parameters are used to
//                adjust copmaring two frames to adjust for
noise.
//
//                In order to better control motion detection,
several concepts are used:
//
//                Segments - A video frame is broken into n
linear segments, so that one segment at a time is
//                checked for motion detection.
If Segments = 1, then the whole frame is checked.
//                If Segments = 2, then half
the frame is checked for motion each interval period.
//                The filter automatically
rotates each segment to be checked in sequence.
//                The default value is 1
segment
//
//                Intervals - The interval is the number of one-
tenth seconds between detecting for motion.
//                The default value is 1
second.(10 intervals)
//
//                Granularity - The granularity is the nth number
of pixels checked during pixel comparisons for
//                detecting motion. In other
words, a granularity of 1 means every pixel is checked,
//                a granularity of 2 means
every second pixel is checked, etc. This value is used to
//                increase performance and
decrease CPU utilization
//                The default value is 4
//
//                Color Sensitivity (Color Error Margin) - This
is the margin of error allowed, particularly for live cameras,
//                when comparing blue with
blue, red with red, etc. to determine whether there is a change, i.e.
//                motion is detected. Since
many live cameras are not precise, a camera might pick up
//                a still shot with a red value
of 120, 121, and 122 in sequential frames, even though
//                nothing has changed in the
view. This is likely due to many factors, including variations
//                in lighting, imprecise camera
hardware, etc.
//                The default value is 6
//
//                % Change Thresholds - Low and high threshold
for determining what constitutes a change in motion. Since

```

```

//                                video is not precise, if
there is a 1% change in pixels between frames it is unlikely to be
//                                a change in motion.
//                                The default value for the low
threshold is 5% (i.e. % pixels changed less than 5% are not considered
motion)
//                                The default value for the
high threshold is 100%
//
//                                Dwell Time - Dwell Time is the amount of time
that the motion detection filter will wait until it checks
//                                to see if there is motion,
from the time it first detects motion. Values are in seconds.
//                                A Dwell Time of 10 means that
the filter will wait 10 seconds from the time it detects motion
//                                until it checks again if
there is any motion. The reason for this is to give time for the
calling
//                                program to react to the
motion detection, rather than keep sending motion detection signals
repeatedly.
//                                The minimum value is 1
second, the maximum is 3600 seconds (1 Hour)
//                                The default value is 60
Seconds
//
//                                Another aspect of this filter is that it lets
you determine the type of motion detection operation as follows:
//
//                                Operation 1: None - Useful if you want to place
filter in a filter graph but shut off the
//                                resource-
intensive motion detection (MDF_VAL_OPERATION_NONE)
//                                m_nOperation =
1006
//                                Operation 2: Standard - Turns on motion
detection as described above
//                                - this is the
default - (MDF_VAL_OPERATION_STANDARD)
//                                m_nOperation =
1007
//                                Operation 3: Fixed - This will send a message
as if there was motion detected every fixed interval
//                                as defined by the
Interval variable - this is very useful for testing purposes
//                                so that you can
see how your application reacts to a steady and predictable
//                                series of "motion
detection" events - (MDF_VAL_OPERATION_FIXED)
//                                m_nOperation =
1008
//
//
//
// Debug Notes: Since debugging live video is very cumbersome, a set of
debug outputs has been coded.

```

```

//          When run in debug mode the debug display will
list all aspects of the video
//          progress (e.g. stream started, interval
reached, etc.), along with critical video frame data (height, width),
and
//          motion detection data when relevant (pixels
viewed, pixels changed, etc.) to help debugging.
//
// Event Notification: When motion is detected an EC_OLE_EVENT message
is sent to the application
//
//
// NOTE:   The default values chosen were selected based on testing
the new video grabber and live video.
//          For pre-recorded video playback, which has far less
noise, the Color Sensitivity and Change
//          Threshold should be reduced.
//
// Limitations:  1)   Does not work on compressed video (since
comparing pixels is irrelevant) but does not check
//                  media subtype to ensure that only
uncompressed video is being used in input pin
//
// Version: 01/21/02   1.00.001   Original - basic - no prop pages,
interface, persistence
//          02/11/02   1.00.002   Changed interval period to
1/10 seconds, and default to 10 intervals (1 sec)
//          02/12/02   1.01.001   Added Property Pages, IID to
variables, and persistence
//          02/13/02   1.01.002   Added type of operation
//          04/23/02   1.01.003   Changed from time stamp to
using clock ticks to determine how far along the video progressed
//                                     The old way I was
checking the pSample time to determine how far along the video
progressed
//                                     However -
apparently Preview mode does not stamp the time on the video and
therefore
//                                     the time stamp is
always zero - therefore I changed to using tick counts from the time
video has
//                                     started. It is a
relative time in milliseconds
//                                     I left the old
code that uses pSample-GetTime to determine how far along the video is
//                                     The only thing
this affects is the IsNextIntervalReached and
//          04/23/02   1.01.004   Added Dwell Time capability.
See description above for Dwell Time variable
//          05/30/02   1.01.005   Changed default dwell time to
60 seconds, and started MDFilter as if motion was detected
//                                     so that there is
a buffer when video starts before motion can be triggered equal to the
//                                     dwell time
//          06/27/02   1.01.006   Changed default dwell time
back to 30 seconds
//

```

```

//
// History: 01/21/02    LCK          Created
//           02/12/02    LCK          Add persistence, property
page and external interface
//           02/13/02    LCK          Added support for type of
operation
//           04/23/02    LCK          See note to 1.01.003
//           04/25/02    LCK          See note to 1.01.004
//           05/30/02    LCK          See note to 1.01.005
//           06/27/02    LCK          See note to 1.01.006
//
//
// Copyright (c) 2002 BKLK Inc. All rights reserved.
//-----
-----

#include "stdio.h"

#include <streams.h>          // DirectShow (includes windows.h)
                              // MAKE SURE CORRECT PATHS
ARE SET IN TOOLS | OPTIONS
                              // OR EMBEDDED #INCLUDES WILL
NOT BE FOUND
                              // THIS VERSION USED INCLUDE
DIRECTORIES FROM THE DIRECTX8.1 SDK
                              // CHECK THIS PROJECT'S
PROJECT SETTINGS FOR EXACT PATHS OF INCLUDE
                              // AND LIB DIRETORIES

#include <initguid.h>         // declares DEFINE_GUID to declare an
EXTERN_C const.
#include "resource.h"
#include "Motion Detection Filter GUIDs.h"
#include "Motion Detection Filter.h"
#include "Motion Detection Filter Prop Page.h"

// See note to 1.01.003
// #define MDFILTER_ONETENTHSECOND 1000000
#define MDFILTER_ONETENTHSECOND 100
#define MDFILTER_ONESECOND 1000

//-----
-----
//
// CMotionDetectionFilter Filter Dscription Data
//
//-----
-----

// Setup data - allows the self-registration to work.
// TODO: Change video subtypes to only allow valid data - i.e.
uncompressed video

const AMOVIESETUP_MEDIATYPE sudPinTypes =
{ &MEDIATYPE_Video          // clsMajorType

```

```

, &MEDIASUBTYPE_RGB24 }; // clsMinorType
//, &MEDIASUBTYPE_NULL }; // clsMinorType

const AMOVIESETUP_PIN psudPins[] =
{ { L"Input" // strName
, FALSE // bRendered
, FALSE // bOutput
, FALSE // bZero
, FALSE // bMany
, &CLSID_NULL // clsConnectsToFilter
, L"" // strConnectsToPin
, 1 // nTypes
, &sudPinTypes // lpTypes
}
, { L"Output" // strName
, FALSE // bRendered
, TRUE // bOutput
, FALSE // bZero
, FALSE // bMany
, &CLSID_NULL // clsConnectsToFilter
, L"" // strConnectsToPin
, 1 // nTypes
, &sudPinTypes // lpTypes
}
};

const AMOVIESETUP_FILTER sudMotionDetection =
{ &CLSID_MotionDetection // clsID
, L"Motion Detection" // strName
, MERIT_DO_NOT_USE // dwMerit
, 2 // nPins
, psudPins }; // lpPin

// Needed for the CreateInstance mechanism
CFactoryTemplate g_Templates[] =
{ { L"Motion Detection"
, &CLSID_MotionDetection
, CMotionDetectionFilter::CreateInstance
, NULL
, &sudMotionDetection }
,
{ L"Main Property Page"
, &CLSID_MDF_PROPPAGEMAIN
, CPropPageMain::CreateInstance }
};
int g_cTemplates = sizeof(g_Templates)/sizeof(g_Templates[0]);

//-----
//
// CMotionDetectionFilter Class Functions
//
//-----

```

```

//
// Constructor
//
// Description: Just calls the base class constructors and
initializes variables
//
CMotionDetectionFilter::CMotionDetectionFilter(TCHAR *tszName,
LPUNKNOWN punk, HRESULT *phr)
    : CTransInPlaceFilter (tszName, punk,
CLSID_MotionDetection, phr)
    , CPersistStream(punk, phr)
{
    m_lpFrameDataBuffer = NULL;
    m_nNumSegments      = MDF_DEF_NUMSEGMENTS;
    m_nSegmentSize      = 0;
    m_nCurrentSegment = 0;
    m_nIntervalPeriod = MDF_DEF_INTERVALPERIOD;
    m_nElapsedIntervals = 0;
    m_nGranularity      = MDF_DEF_GRANULARITY;
    m_nPercentThresholdLow = MDF_DEF_PERCENTTHRESHOLDLOW;
    m_nPercentThresholdHigh = MDF_DEF_PERCENTTHRESHOLDHIGH;
    m_nColorErrorMargin = MDF_DEF_COLORERRORMARGIN;
    m_nOperation        = MDF_DEF_OPERATION;
    m_dwMDTime          = NULL;
    m_nDwellTime        = MDF_DEF_DWELLTIME;
    m_dwStartTime       = NULL;
} // Constructor

//
// CreateInstance
//
// Description: Provide the way for COM to create a
CMotionDetectionFilter object
//
CUnknown * WINAPI CMotionDetectionFilter::CreateInstance(LPUNKNOWN
punk, HRESULT *phr) {

    CMotionDetectionFilter *pNewObject = new
CMotionDetectionFilter(NAME("Motion Detection In-Place Transform
Filter"), punk, phr );
    if (pNewObject == NULL) {
        *phr = E_OUTOFMEMORY;
    }

    return pNewObject;
} // CreateInstance

//
// Transform
//
// Description: Override Transform base class function to view frames
one at a time and determine whether motion has
been detected.
//
Motion is detected when there is a pixel change
between frames.

```

```

//
HRESULT CMotionDetectionFilter::Transform(IMediaSample *pSample)
{
    HRESULT t_hResult = S_OK;

    // Determine which operation to perform and call appropriate
function
    switch ( m_nOperation )
    {
    case MDF_VAL_OPERATION_STANDARD:
        t_hResult = StandardOperation(pSample);
        break;
    case MDF_VAL_OPERATION_FIXED:
        t_hResult = FixedIntervalOperation(pSample);
        break;
    case MDF_VAL_OPERATION_NONE:
    default:
        break;
    }

    return t_hResult;
} // Transform


//
// StartStreaming
//
// Description:  Override StartStreaming base class function to
initialize variables
//                used for detecting motion for a new stream.
//
//                The main purpose of this function is to
initialize a buffer of sufficient
//                size to save one frame fo video data. This will
be used to compare one frame
//                with a subsequent frame.
//
//                Additionally, several variables are initialized
to zero states.
//
HRESULT CMotionDetectionFilter::StartStreaming(void)
{
    AM_MEDIA_TYPE* t_pType = &m_pInput->CurrentMediaType();
    VIDEOINFOHEADER *t_pvi = (VIDEOINFOHEADER *) t_pType->pbFormat;

    // STEP 1:  Get the size of buffer necessary to contain a single
frame of video
    //                Get the image properties from the
BITMAPINFOHEADER for each frame
    //                Note that the image size does not necessarily
equate to
    //                the amount of bytes of storage. Each image
"pixel" can
    //                actually be multiple bytes of information (e.g.
3 bytes

```

```

//                                one each for red, blue, and green)

int t_cxImage   = t_pvi->bmiHeader.biWidth;      // Image width
int t_cyImage   = t_pvi->bmiHeader.biHeight;     // Image height
int t_nPixels   = t_cxImage * t_cyImage;         // Image size in
video pixels

// Get the actual amount of storage space required for each frame
// which may differ (depending on the size of each pixel of video

int t_nPixelSize = t_pvi->bmiHeader.biBitCount / 8;      //
bytes/pixel of video
int t_nImageSize = t_cyImage * t_cxImage * t_nPixelSize; //
Total max number of bytes to keep a

// single frame of video data

// STEP 2: Allocate the buffer
//          The buffer will be deallocated in StopStreaming
when the video stream ends
//          The buffer can be re-used since we only check
one frame at a time
//          This is more efficient than allocating the
buffer when it is needed for each frame check

if ( m_lpFrameDataBuffer != NULL )
{
    // This should be impossible to reach, but just in case...

    delete m_lpFrameDataBuffer;
    m_lpFrameDataBuffer = NULL;
}

m_lpFrameDataBuffer = new BYTE[t_nImageSize];

// STEP 3: Initialize other variables necessary for motion
detection
//          See intro comments for more info on variable
use

m_nSegmentSize = t_nPixels/m_nNumSegments;           // Segment
size to be checked (in video pixels)
m_nCurrentSegment = 0;                                //
Reset cuurent segment to 0
m_nElapsedIntervals = 0;                              //
Start of stream - 0 elapsed intervals

#ifdef DEBUG
char t_szDebugBuffer[512];

DbgOutString("Start Streaming Variables\n");
sprintf(t_szDebugBuffer, "Video Frame Size: \
    Pix Width = %d, \
    Pix Height = %d, \
    Tot Pixels = %d, \
    Bytes per pixel = %d, \

```

```

        Image Size (bytes) = %d, \
        Segment Size (pixels) = %d \n", \
        t_cxImage, t_cyImage, t_nPixels,
t_nPixelSize, t_nImageSize, m_nSegmentSize);
    DbgOutString(t_szDebugBuffer);
#endif

    // STEP 4: Set the start time
    m_dwStartTime = GetTickCount();

    // 1.01.005 - Start as if motion was detected to prevent it from
    firing immediately
    // m_dwMDTime = GetTickCount();
    m_dwMDTime = NULL;

    return S_OK;
} // StartStreaming

//
// StopStreaming
//
// Description:  Override StopStreaming base classfunction to clean
up variables
//              used for detecting motion for a new stream.
//
HRESULT CMotionDetectionFilter::StopStreaming(void)
{
    // STEP 1: Deallocate buffer used to contain video frame for
    motion detection

    if ( m_lpFrameDataBuffer != NULL )
    {
        delete m_lpFrameDataBuffer ;
        m_lpFrameDataBuffer = NULL;
    }

#ifdef DEBUG
    DbgOutString("Stop Streaming\n");
#endif

    return S_OK;
} // StopStreaming

//
// CheckInputType
//
// Override default CheckInputType function
// Check the input type is OK - return an error otherwise
//
HRESULT CMotionDetectionFilter::CheckInputType(const CMediaType *mtIn)
{
    // check this is a VIDEOINFOHEADER type

```

```

    if (*mtIn->FormatType() != FORMAT_VideoInfo) {
        return E_INVALIDARG;
    }

    // Can we transform this type

    if (CanPerformRGB24(mtIn)) {
        return NOERROR;
    }
    return E_FAIL;
}

//
// Checktransform
//
// Override default Checktransformfunction
// Check a transform can be done between these formats
//
HRESULT CMotionDetectionFilter::CheckTransform(const CMediaType *mtIn,
const CMediaType *mtOut)
{
    if (CanPerformRGB24(mtIn)) {
        if (*mtIn == *mtOut) {
            return NOERROR;
        }
    }
    return E_FAIL;
} // CheckTransform

// CanPerformEZrgb24
//

// Check if this is a RGB24 true colour format
//
BOOL CMotionDetectionFilter::CanPerformRGB24(const CMediaType
*pMediaType) const
{
    if (IsEqualGUID(*pMediaType->Type(), MEDIATYPE_Video)) {
        if (IsEqualGUID(*pMediaType->Subtype(), MEDIASUBTYPE_RGB24)) {
            VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) pMediaType->
>Format();
            return (pvi->bmiHeader.biBitCount == 24);
        }
    }
    return FALSE;
} // CanPerformEZrgb24

//
// NonDelegatingQueryInterface
//
// Reveals ISpecifyPropertyPages and other interfaces
//

```

```

STDMETHODIMP CMotionDetectionFilter::NonDelegatingQueryInterface(REFIID
riid, void **ppv)
{
    CheckPointer(ppv, E_POINTER);

    if (riid == IID_MDF_CONTROLVARIABLES)
    {
        return GetInterface((IIPMDFControlVariables*) this, ppv);
    }
    else if (riid == IID_ISpecifyPropertyPages)
    {
        return GetInterface((ISpecifyPropertyPages *) this, ppv);
    }
    else if (riid == IID_IPersistStream)
    {
        return GetInterface((IPersistStream *) this, ppv);
    }
    else
    {
        return CTransInPlaceFilter::NonDelegatingQueryInterface(riid,
ppv);
    }
} // NonDelegatingQueryInterface


//
// StandardOperation
//
// Description:  Motion is detected when there is a pixel change
between frames.
//
HRESULT CMotionDetectionFilter::StandardOperation(IMediaSample
*pSample)
{
    static bool t_bCompareFrame = false; // Compare frame to last
frame
    BYTE *t_pData; // Pointer to the actual
image buffer
    RGBTRIPLE *t_pr gbLastFrame; // Holds a pointer to the current pixel
of the last frame
    RGBTRIPLE *t_pr gbThisFrame; // Holds a pointer to the current pixel
of this frame
    int t_nPixel; // Used to loop through the image
pixels
    int t_nDifferent; // Number of pixels that chanded
between frames
    int t_nPercentChange; // % of pixel change in segment
being examined
    int t_nPixelsCompared; // Actual number of pixels
compared, which may be different
// than the segment
size if the granularity is not 1


    // STEP 1: Get the pointer to the actual image buffer of the
current frame,

```

```

//                                and other key data of the current video frame

pSample->GetPointer(&t_pData); // Get pointer to the actual data
of the sample

// STEP 2: Check if it is time for the next motion detection and
if so
//                                copy the current frame into the temp buffer to
compare it to                        the next frame.
//                                Set the compare frame variable to true
//                                then exit - nothing more to do until next
frame

    if ( IsNextIntervalReached(pSample) )
    {
#ifdef DEBUG
        DbgOutString("Copy frame\n");
#endif
        // We use ActualDataLength as a safeguard in case the
        // current frame is using less
        // data than the max (this would happen in compressed
        // formats)
        long t_lActualDataLength = pSample->GetActualDataLength();
        CopyMemory( (LPVOID) m_lpFrameDataBuffer, (LPVOID) t_pData,
t_lActualDataLength );
        t_bCompareFrame = true;
        goto Exit1;
    }

// STEP 3: Check if this frame needs to be compared to the last
frame
//                                i.e. if the last frame reached the check
interval, and if not
//                                simply exit. If this frame needs to be checked,
continue and
//                                reset comparison variable

    if ( !t_bCompareFrame )
        goto Exit1;
    t_bCompareFrame = false;

#ifdef DEBUG
    DbgOutString("Compare frame\n");
#endif

// STEP 4: Set up the data buffers for the pixel comparison

    t_pr gbThisFrame = (RGBTRIPLE*) t_pData;
    t_pr gbLastFrame = (RGBTRIPLE*) m_lpFrameDataBuffer;
    t_pr gbThisFrame = &t_pr gbThisFrame[m_nCurrentSegment *
m_nSegmentSize];
    t_pr gbLastFrame = &t_pr gbLastFrame[m_nCurrentSegment *
m_nSegmentSize];

```

```

        // STEP 5: Detect motion by comparing pixels from last frame
        buffer with
        //          pix ls in this frame buffer, and note the
        number that changed

        for (t_nPixelsCompared = 0, t_nPixel = 0, t_nDifferent = 0;
        t_nPixel < m_nSegmentSize; \
            t_nPixel++, t_pr gbLastFrame++, t_pr gbThisFrame++)
        {
            // Adjust for granularity by only comparing the nth
            (m_nGranularity) pixel
            // and skipping the rest

            div_t t_divResult;
            t_divResult = div(t_nPixel, m_nGranularity);
            if ( t_divResult.rem != 0 )
                continue;

            // Increment the pixel comparison count
            t_nPixelsCompared++;

            // Compare last frame's pixels with this frame's pixel and
            increment the number
            if ( abs(t_pr gbThisFrame->rgbtBlue - t_pr gbLastFrame-
            >rgbtBlue ) > m_nColorErrorMargin
                && abs(t_pr gbThisFrame->rgbtGreen - t_pr gbLastFrame-
            >rgbtGreen ) > m_nColorErrorMargin
                && abs(t_pr gbThisFrame->rgbtRed - t_pr gbLastFrame-
            >rgbtRed ) > m_nColorErrorMargin
            )
            {
                t_nDifferent++;
            }
        }

        // STEP 6: Calc % change, determine whether there was motion,
        and send event to
        //          Filter Graph as appropriate

        // Calc % change
        t_nPercentChange = t_nDifferent * 100 / t_nPixelsCompared;

        // Determine if motion is detected and if so send event to
        filter graph
        if ( t_nPercentChange > m_nPercentThresholdLow &&
        t_nPercentChange < m_nPercentThresholdHigh )
        {
            static BSTR t_bstrMessage;
            static WCHAR * t_wszMessage = L"MDFilter";
            WriteBSTR(&t_bstrMessage, t_wszMessage);
            NotifyEvent(EC_OLE_EVENT, (LONG_PTR) t_bstrMessage,
            (LONG_PTR) NULL);
            // TODO: look into whether the app frees this - I think so
            because otherwise there is a mem heap alloc error
            //          FreeBSTR(&t_bstrMessage);

```

```

        // Set the time when motion was detected so that we can
avoid sending motion detection notifications
        // during the dwell time
        m_dwMDTime = GetTickCount();

#ifdef DEBUG
        DbgOutString("Motion detection event notification sent\n");
#endif
    }

#ifdef DEBUG
        char t_szDebugBuffer[256];
        sprintf(t_szDebugBuffer, "Segment = %d, Seg Size = %d, Pix
Compared = %d, Pix Diff = %d, %% diff = %d%%\n", \
                m_nCurrentSegment, t_nPixel,
t_nPixelsCompared, t_nDifferent, t_nPercentChange );
        DbgOutString(t_szDebugBuffer);
#endif

        // STEP 7: Increment current segment, rotate back to 0 if all
segments were checked

        m_nCurrentSegment++;
        if (m_nCurrentSegment == m_nNumSegments ) m_nCurrentSegment = 0;

Exit1:
    return S_OK;
} // StandardOperation

//
// FixedIntervalOperation
//
// Description: Motion event is set at fixed intervals set by the
interval variable
//
HRESULT CMotionDetectionFilter::FixedIntervalOperation(IMediaSample
*pSample)
{
    // STEP 1: Check if it is time for the interval and if so
    // copy the current frame into the temp buffer to
compare it to
    // the next frame.
    // Set the compare frame variable to true
    // then exit - nothing more to do until next
frame

    if ( IsNextIntervalReached(pSample) )
    {
        static BSTR t_bstrMessage;
        static WCHAR * t_wszMessage = L"MDFilter";
        WriteBSTR(&t_bstrMessage, t_wszMessage);
        NotifyEvent(EC_OLE_EVENT, (LONG_PTR) t_bstrMessage,
(LONG_PTR) NULL);
    }
}

```

```

        // TODO: look into whether the app frees this - I think so
        because otherwise there is a mem heap alloc error
        //      FreeBSTR(&t_bstrMessage);
    #if DEBUG
        DbgOutString("Fixed interval event notification sent\n");
    #endif
    }

    return S_OK;
} // FixedIntervalOperation

//
// IsNextIntervalReached
//
// Description:  Determines whether next interval has been reached to
// check for motion change
//
bool CMotionDetectionFilter::IsNextIntervalReached(IMediaSample
*pSample)
{
    bool t_bReturn = false;                // Assume next interval
has not been reached
    //      CRefTime t_tStart, t_tStop ;      // Start and stop times of
the frame (in nanoseconds)
    long t_lNextIntervalTime = 0;          // Next time motion needs to
be detected
    DWORD t_dwCurrentTime;
    DWORD t_dwDwellTimeOver;               // If motion detected,
time when Dwell Time is completed

    // STEP 1: Calculate next interval time that motion needs to be
detected (in nanoseconds)

    // See note to 1.01.003
    //      t_lNextIntervalTime = ((long) m_nElapsedIntervals * (long)
m_nIntervalPeriod + (long) m_nIntervalPeriod) * (long)
MDFILTER_ONETENTHSECOND;
    t_lNextIntervalTime = (long) m_dwStartTime + ((long)
m_nElapsedIntervals * (long) m_nIntervalPeriod + (long)
m_nIntervalPeriod) * (long) MDFILTER_ONETENTHSECOND);

    // STEP 2: Get current frame times

    // See note to 1.01.003
    //      pSample->GetTime((REFERENCE_TIME *) &t_tStart, (REFERENCE_TIME *)
&t_tStop);
    t_dwCurrentTime = GetTickCount();

    #if DEBUG
        char t_szDebugBuffer[256];
        long t_lDataLen = pSample->GetSize();
        long t_lActualDataLength = pSample->GetActualDataLength();
        //      sprintf(t_szDebugBuffer, "AStart = %ld, Stop = %ld, Next Interval
Time = %ld, Data Size = %ld, Actual Data Length = %ld\n", (long)

```

```

t_tStart.m_time, (long) t_tStop.m_time, t_lNextIntervalTime,
t_lDataLen, t_lActualDataLength );
    sprintf(t_szDebugBuffer, "Current Time = %ld, Next Interval Time
= %ld, Data Size = %ld, Actual Data Length = %ld\n", (long)
t_dwCurrentTime, t_lNextIntervalTime, t_lDataLen, t_lActualDataLength
);
    DbgOutString(t_szDebugBuffer);
#endif

    // STEP 3: If motion was already detected, make sure that the
    dwell time has passed
    // See note to 1.00.004

    if ( m_dwMDTime > 0 )
    {
        // Dwell time is over dwell time * seconds from the motion
        detection time
        t_dwDwellTimeOver = m_dwMDTime + (MDFILTER_ONESECOND *
m_nDwellTime );
        if ( t_dwCurrentTime < t_dwDwellTimeOver )
        {
            // Dwell time is not yet over - exit false
#ifdef DEBUG
            DbgOutString("In dwell time - dwell time not yet over\n");
#endif
            goto Exit1;
        }

        // If the dwell time is over, reset the Motion Detection
        Time back to zero
        m_dwMDTime = 0;
    }

    // STEP 4: Determine whether current time reached next interval
    time
    // and if so, increment next interval time and set
    return value
    // to true

    // See note to 1.01.003
    // if ( t_tStart.m_time >= t_lNextIntervalTime )
    if ( (long) t_dwCurrentTime >= t_lNextIntervalTime )
    {
        m_nElapsedIntervals++;
        t_bReturn = true;
#ifdef DEBUG
        sprintf(t_szDebugBuffer, "Elapsed intervals: %d\n",
m_nElapsedIntervals );
        DbgOutString(t_szDebugBuffer);
#endif
    }

Exit1:
    return t_bReturn;
} // IsNextIntervalReached

```

```

//*****
// ISpecifyPropertyPages override functions necessary for
property pages
//*****

// GetPages
//
// Returns the clsid's of the property pages we support
//
STDMETHODIMP CMotionDetectionFilter::GetPages(CAUUID *pPages)
{
    pPages->cElems = 1;
    pPages->pElems = (GUID *) CoTaskMemAlloc(sizeof(GUID));
    if (pPages->pElems == NULL) {
        return E_OUTOFMEMORY;
    }
    *(pPages->pElems) = CLSID_MDF_PROPPAGEMAIN;
    return NOERROR;
} // GetPages

//*****
// IIPMDFControlVariables override functions for transferring
variables
//*****

//
// get_NumSegments
//
// Returns the number of segments control variable
//
STDMETHODIMP CMotionDetectionFilter::get_NumSegments(int *NumSegments)
{
    *NumSegments = m_nNumSegments;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_NumSegments

//
// put_NumSegments
//
// Sets the number of segments control variable
//
STDMETHODIMP CMotionDetectionFilter::put_NumSegments(int NumSegments)
{

```

```

    m_nNumSegments = NumSegments;

    // Check and reset in case requested value is out of bounds
    if ( m_nNumSegments < MDF_MIN_NUMSEGMENTS ) m_nNumSegments =
MDF_MIN_NUMSEGMENTS ;
    if ( m_nNumSegments > MDF_MAX_NUMSEGMENTS ) m_nNumSegments =
MDF_MAX_NUMSEGMENTS ;

    return NOERROR;
} // put_NumSegments

//
// get_IntervalPeriod
//

// Returns the interval period control variable
//
STDMETHODIMP CMotionDetectionFilter::get_IntervalPeriod(int
*IntervalPeriod)
{
    *IntervalPeriod = m_nIntervalPeriod;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_IntervalPeriod

//
// put_IntervalPeriod
//
// Sets the interval period control variable
//
STDMETHODIMP CMotionDetectionFilter::put_IntervalPeriod(int
IntervalPeriod)
{
    m_nIntervalPeriod = IntervalPeriod;

    // Check and reset in case requested value is out of bounds
    if ( m_nIntervalPeriod < MDF_MIN_INTERVALPERIOD )
m_nIntervalPeriod = MDF_MIN_INTERVALPERIOD ;
    if ( m_nIntervalPeriod > MDF_MAX_INTERVALPERIOD )
m_nIntervalPeriod = MDF_MAX_INTERVALPERIOD ;

    return NOERROR;
} // put_IntervalPeriod

//
// get_Granularity
//

```

```

// Returns the granularity control variable
//
STDMETHODIMP CMotionDetectionFilter::get_Granularity(int *Granularity)
{
    *Granularity = m_nGranularity;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_Granularity

//
// put_Granularity
//
// Sets the granularity control variable
//
STDMETHODIMP CMotionDetectionFilter::put_Granularity(int Granularity)
{
    m_nGranularity = Granularity;

    // Check and reset in case requested value is out of bounds
    if ( m_nGranularity < MDF_MIN_GRANULARITY ) m_nGranularity =
MDF_MIN_GRANULARITY;
    if ( m_nGranularity > MDF_MAX_GRANULARITY ) m_nGranularity =
MDF_MAX_GRANULARITY;

    return NOERROR;
} // put_Granularity

//
// get_ColorErrorMargin
//
// Returns the color error margin control variable
//
STDMETHODIMP CMotionDetectionFilter::get_ColorErrorMargin(int
*ColorErrorMargin)
{
    *ColorErrorMargin = m_nColorErrorMargin;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_ColorErrorMargin

//
// put_ColorErrorMargin
//
// Sets the color error margin control variable

```

```

//
STDMETHODIMP CMotionDetectionFilter::put_ColorErrorMargin(int
ColorErrorMargin)
{
    m_nColorErrorMargin = ColorErrorMargin;

    // Check and reset in case requested value is out of bounds
    if ( m_nColorErrorMargin < MDF_MIN_COLORERRORMARGIN )
m_nColorErrorMargin = MDF_MIN_COLORERRORMARGIN ;
        if ( m_nColorErrorMargin > MDF_MAX_COLORERRORMARGIN )
m_nColorErrorMargin = MDF_MAX_COLORERRORMARGIN;

    return NOERROR;
} // put_ColorErrorMargin

//
// get_PercentThresholdHigh
//
// Returns the percent threshold high control variable
//
STDMETHODIMP CMotionDetectionFilter::get_PercentThresholdHigh(int
*PercentThresholdHigh)
{
    *PercentThresholdHigh = m_nPercentThresholdHigh;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_PercentThresholdHigh

//
// put_PercentThresholdHigh
//
// Sets the percent threshold high control variable
//
STDMETHODIMP CMotionDetectionFilter::put_PercentThresholdHigh(int
PercentThresholdHigh)
{
    m_nPercentThresholdHigh = PercentThresholdHigh;

    // Check and reset in case requested value is out of bounds
    if ( m_nPercentThresholdHigh < MDF_MIN_PERCENTTHRESHOLDHIGH )
m_nPercentThresholdHigh = MDF_MIN_PERCENTTHRESHOLDHIGH;
        if ( m_nPercentThresholdHigh > MDF_MAX_PERCENTTHRESHOLDHIGH )
m_nPercentThresholdHigh = MDF_MAX_PERCENTTHRESHOLDHIGH;

    return NOERROR;
} // put_PercentThresholdHigh

//

```

```

// get_PercentThresholdLow
//
// Returns the percent threshold low control variable
//
STDMETHODIMP CMotionDetectionFilter::get_PercentThresholdLow(int
*PercentThresholdLow)
{
    *PercentThresholdLow = m_nPercentThresholdLow;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_PercentThresholdLow


//
// put_PercentThresholdLow
//
// Sets the percent threshold low control variable
//
STDMETHODIMP CMotionDetectionFilter::put_PercentThresholdLow(int
PercentThresholdLow)
{
    m_nPercentThresholdLow = PercentThresholdLow;

    // Check and reset in case requested value is out of bounds
    if ( m_nPercentThresholdLow < MDF_MIN_PERCENTTHRESHOLDLOW)
        m_nPercentThresholdLow = MDF_MIN_PERCENTTHRESHOLDLOW;
    if ( m_nPercentThresholdLow > MDF_MAX_PERCENTTHRESHOLDLOW)
        m_nPercentThresholdLow = MDF_MAX_PERCENTTHRESHOLDLOW;

    return NOERROR;
} // put_PercentThresholdLow


//
// get_Operation
//
// Returns the operation variable
//
STDMETHODIMP CMotionDetectionFilter::get_Operation(int *Operation)
{
    *Operation = m_nOperation;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_Operation


//
// put_Operation

```

```

//
// Sets the operation variable
//
STDMETHODIMP CMotionDetectionFilter::put_Operation(int Operation)
{
    m_nOperation = Operation;

    // Check and reset in case requested value is out of bounds
    if ( m_nOperation < IDC_OPNONE ) m_nOperation = IDC_OPNONE ;
    if ( m_nOperation > IDC_OPFIXED ) m_nOperation = IDC_OPFIXED ;

    return NOERROR;
} // put_Operation

//
// get_DwellTime
//
// Returns the Dwell Time variable
//
STDMETHODIMP CMotionDetectionFilter::get_DwellTime(int *DwellTime)
{
    *DwellTime = m_nDwellTime;

    // Set flag to stream value
    CPersistStream::SetDirty(TRUE);

    return NOERROR;
} // get_Operation

//
// put_DwellTime
//
// Sets the Dwell Time variable
//
STDMETHODIMP CMotionDetectionFilter::put_DwellTime(int DwellTime)
{
    m_nDwellTime = DwellTime ;

    // Check and reset in case requested value is out of bounds
    if ( m_nDwellTime < MDF_MIN_DWELLTIME ) m_nDwellTime =
MDF_MIN_DWELLTIME;
    if ( m_nDwellTime > MDF_MAX_DWELLTIME ) m_nDwellTime =
MDF_MAX_DWELLTIME;

    return NOERROR;
} // put_Operation

//*****
*****

```

```

        // CPersistStream override functions necessary for streaming
        //*****
*****

//
// SizeMax
//
// Override CPersistStream method.

// State the maximum number of bytes we would ever write in a file
// to save our properties.
//
int CMotionDetectionFilter::SizeMax()
{
    // When an int is expanded as characters it takes at most 12
    characters
    // including a trailing delimiter.
    // Wide chars doubles this and we want six ints.
    //
    return 256; // very safe!
} // SizeMax

//
// WriteToStream
//
// Override CPersistStream method.
// Write our properties to the stream.
//
HRESULT CMotionDetectionFilter::WriteToStream(IStream *pStream)
{
    HRESULT hr;

    hr = WriteInt(pStream, m_nNumSegments);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nIntervalPeriod);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nGranularity);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nColorErrorMargin);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nPercentThresholdHigh);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nPercentThresholdLow);
    if (FAILED(hr)) return hr;

    hr = WriteInt(pStream, m_nOperation);
    if (FAILED(hr)) return hr;

    return NOERROR;
} // WriteToStream

```

```

//
// ReadFromStream
//
// Override CPersistStream method.
// Read our properties from the stream.
//
HRESULT CMotionDetectionFilter::ReadFromStream(IStream *pStream)
{
    HRESULT hr;

    m_nNumSegments = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nIntervalPeriod = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nGranularity = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nColorErrorMargin = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nPercentThresholdHigh = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nPercentThresholdLow = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    m_nOperation = ReadInt(pStream, hr);
    if (FAILED(hr)) return hr;

    return NOERROR;
} // ReadFromStream

// GetClassID
//
// Override CBaseMediaFilter method for interface IPersist
// Part of the persistent file support. We must supply our class id
// which can be saved in a graph file and used on loading a graph with
// a Motion Detection in it to instantiate this filter via
// CoCreateInstance.
//
STDMETHODIMP CMotionDetectionFilter::GetClassID(CLSID *pClsid)
{
    if (pClsid==NULL) {
        return E_POINTER;
    }
    *pClsid = CLSID_MotionDetection;
    return NOERROR;
} // GetClassID

```

```

/*****Public*Routine*****/
****\
* exported entry points for registration and
* unregistration (in this case they only call
* through to default implementations).
*
*
*
* History:
*
\*****/
STDAPI
DllRegisterServer()
{
    return AMovieDllRegisterServer2( TRUE );
}

STDAPI
DllUnregisterServer()
{
    return AMovieDllRegisterServer2( FALSE );
}

// Microsoft C Compiler will give hundreds of warnings about
// unused inline functions in header files. Try to disable them.
#pragma warning( disable:4514)
//-----
//
// Class: CMotionDetectionFilter
//
// Description: In-place transfer filter to view pixels of video frames
//              and determine whether there is any motion
// change
//
// History: 01/21/02    LCK          Created
//          02/12/02    LCK          Add persistence, property
//          02/13/02    LCK          Added support for type of
//          operation
//
//-----

// Default initial values for controlling variables

#define MDF_DEF_NUMSEGMENTS          1
#define MDF_DEF_INTERVALPERIOD      3
#define MDF_DEF_GRANULARITY         4
#define MDF_DEF_PERCENTTHRESHOLDLOW 10
#define MDF_DEF_PERCENTTHRESHOLDHIGH 99
#define MDF_DEF_COLORERRORMARGIN    6
#define MDF_VAL_OPERATION_NONE      IDC_OPNONE
#define MDF_VAL_OPERATION_STANDARD  IDC_OPSTANDARD
#define MDF_VAL_OPERATION_FIXED     IDC_OPFIXED
#define MDF_DEF_OPERATION           MDF_VAL_OPERATION_STANDARD

```

```

#define MDF_DEF_DWELLTIME          30

class CMotionDetectionFilter
    : public CTransInPlaceFilter
    , public IIPMDFControlVariables // Extenal interface for
transferring controlling variables
    , public ISpecifyPropertyPages // Needed for property pages
    , public CPersistStream        // Implements IPersistStream to
save values with filter graph
{
    //*****
    // Minimal CTransInPlaceFilter Override functions of base class
    // These are the minimal functions that need to be declared to
run a
    // DirectShow Transform filter
    //*****
public:
    // CreateInstance
    static CUnknown *WINAPI CreateInstance(LPUNKNOWN punk, HRESULT
*phr);

    DECLARE_IUNKNOWN;
private:
    // Constructor - just calls the base class constructor and
initializes variables
    CMotionDetectionFilter(TCHAR *tszName, LPUNKNOWN punk, HRESULT
*phr);

    // Overrides the PURE virtual Transform of CTransInPlaceFilter base
class
    // This is where the "real work" is done by altering *pSample.
    HRESULT Transform(IMediaSample *pSample);

    // Override these functions to accept only RGB24
    HRESULT CheckInputType(const CMediaType *mtIn);
    HRESULT CheckTransform(const CMediaType *mtIn, const CMediaType
*mtOut);
    // Helper function for checking that pin is RGB24
    BOOL CanPerformRGB24(const CMediaType *pMediaType) const;

    //*****
    // Additional CTransInPlaceFilter override functions necessary
for this filter
    // (but not necessary for a minimal filter with no interface or
prop pages)
    //*****
private:
    // Pre- and post- stream functions that are used to initialize

```

```

// variables necessary for motion detection
HRESULT StartStreaming(void);
HRESULT StopStreaming(void);

// Reveals property page and other interfaces
STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void ** ppv);

//*****
// ISpecifyPropertyPages override functions necessary for
property pages
//*****

// Gets the property pages when user requests on Filter Properties
STDMETHODIMP GetPages(CAUUID *pPages);

//*****
// CPersistStream override functions necessary for streaming
//*****

HRESULT WriteToStream(IStream *pStream);
HRESULT ReadFromStream(IStream *pStream);
int SizeMax();
STDMETHODIMP GetClassID(CLSID *pClsid);

//*****
// IIPMDFControlVariables override functions for transferring
variables
//*****
STDMETHODIMP get_NumSegments(int *NumSegments);
STDMETHODIMP put_NumSegments(int NumSegments);
STDMETHODIMP get_IntervalPeriod(int *IntervalPeriod);
STDMETHODIMP put_IntervalPeriod(int IntervalPeriod);
STDMETHODIMP get_Granularity(int *Granularity);
STDMETHODIMP put_Granularity(int Granularity);
STDMETHODIMP get_ColorErrorMargin(int *ColorErrorMargin);
STDMETHODIMP put_ColorErrorMargin(int ColorErrorMargin);
STDMETHODIMP get_PercentThresholdHigh(int *PercentThresholdHigh);
STDMETHODIMP put_PercentThresholdHigh(int PercentThresholdHigh);
STDMETHODIMP get_PercentThresholdLow(int *PercentThresholdLow);
STDMETHODIMP put_PercentThresholdLow(int PercentThresholdLow);
STDMETHODIMP get_Operation(int *Operation);
STDMETHODIMP put_Operation(int Operation);
STDMETHODIMP get_DwellTime(int *Operation);
STDMETHODIMP put_DwellTime(int Operation);

//*****

```

```

// Private internal variables and functions
//*****
private:

// Variables and functions necessary for motion detection

LPVOID      m_lpFrameDataBuffer;          // Buffer to hold a
single frame of video
int          m_nNumSegments;               // The number of
segments that the video frame will be divided into
int          m_nSegmentSize;              // The size of
each segment (in video pixels - NOT bytes)
int          m_nCurrentSegment;           // The current
segment being checked, which rotates sequentially
int          m_nIntervalPeriod;           // Number of
seconds between checking for motion changes
int          m_nElapsedIntervals;         // Number of intervals
elapsed since current stream began
int          m_nGranularity;              // Density of
pixels checked for motion change
int          m_nPercentThresholdHigh;     // High threshold %
change above which motion detection is invalid
int          m_nPercentThresholdLow;      // Low threshold %
change above which motion detection is invalid
int          m_nColorErrorMargin;         // Error is the amount
of color differential allowed when comparing green, blue or red

int          m_nOperation;                // Which
operation user selected

// See note to 1.00.003
DWORD m_dwStartTime;                      // Start time of video
- in computer ticks
// See note to 1.00.004
DWORD m_dwMDTime;                         // Time when motion was
detected
int          m_nDwellTime;                // Amount of time
between motion detection notifications

bool IsNextIntervalReached(IMediaSample *pSample);
HRESULT StandardOperation(IMediaSample *pSample);
HRESULT FixedIntervalOperation(IMediaSample *pSample);
};

//-----
// File: Motion Detection Filter GUIDs.h
//
// Desc: Motion Detection DirectShow filter GUIDs
//
// History: 01/21/02   LCK           Created CLSID_MotionDetection
//             02/12/02   LCK           Created
//
//
// Copyright (c) 2002 BKLK Inc. All rights reserved.

```

```

//-----
-----

#ifndef __MDFGUIDS__
#define __MDFGUIDS__

#ifdef __cplusplus
extern "C" {
#endif

// The CLSID used by the Motion Detection Filter
// {236D037F-B012-4b21-A89E-FB1D59130142}
DEFINE_GUID(CLSID_MotionDetection,
0x236d037f, 0xb012, 0x4b21, 0xa8, 0x9e, 0xfb, 0x1d, 0x59, 0x13, 0x1,
0x42);

// The CLSID of the main property page
// {C29DBB7B-E8CE-4ebe-B7B4-B45BD700BA31}
DEFINE_GUID(CLSID_MDF_PROPPAGEMAIN,
0xc29dbb7b, 0xe8ce, 0x4ebe, 0xb7, 0xb4, 0xb4, 0x5b, 0xd7, 0x0, 0xba,
0x31);

// Interface to get/put main motion detection variables from/to filter
// {2E16C296-68E6-45b6-B220-F3E9FAC5D264}
DEFINE_GUID(IID_MDF_CONTROLVARIABLES,
0x2e16c296, 0x68e6, 0x45b6, 0xb2, 0x20, 0xf3, 0xe9, 0xfa, 0xc5, 0xd2,
0x64);

DECLARE_INTERFACE_(IIPMDFControlVariables, IUnknown)
{
    STDMETHOD(get_NumSegments) (THIS_ int *NumSegments) PURE;
    STDMETHOD(put_NumSegments) (THIS_ int NumSegments) PURE;

    STDMETHOD(get_IntervalPeriod) (THIS_ int *IntervalPeriod) PURE;
    STDMETHOD(put_IntervalPeriod) (THIS_ int IntervalPeriod) PURE;

    STDMETHOD(get_Granularity) (THIS_ int *Granularity) PURE;
    STDMETHOD(put_Granularity) (THIS_ int Granularity) PURE;

    STDMETHOD(get_ColorErrorMargin) (THIS_ int *ColorErrorMargin)
PURE;
    STDMETHOD(put_ColorErrorMargin) (THIS_ int ColorErrorMargin)
PURE;

    STDMETHOD(get_PercentThresholdHigh) (THIS_ int
*PercentThresholdHigh) PURE;
    STDMETHOD(put_PercentThresholdHigh) (THIS_ int
PercentThresholdHigh) PURE;

    STDMETHOD(get_PercentThresholdLow) (THIS_ int
*PercentThresholdLow) PURE;
    STDMETHOD(put_PercentThresholdLow) (THIS_ int
PercentThresholdLow) PURE;

    STDMETHOD(get_Operation) (THIS_ int *Operation) PURE;
    STDMETHOD(put_Operation) (THIS_ int Operation) PURE;

```

```
        STDMETHOD(get_DwellTime) (THIS_ int *DwellTime) PURE;  
        STDMETHOD(put_DwellTime) (THIS_ int DwellTime) PURE;  
    };  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif // __MDFGUIDS__
```

```

// VPDApp.h : Declaration of the CVPDApp

class CVPDApp : public CWinApp
{
public:

    // Overrides

        // ClassWizard generated virtual function overrides
        //{AFX_VIRTUAL(CVPDApp)
        public:
            virtual BOOL InitInstance();
            virtual int ExitInstance();
        //}AFX_VIRTUAL

        //{AFX_MSG(CVPDApp)
        // NOTE - the ClassWizard will add and remove member
functions here.        // DO NOT EDIT what you see in these blocks of
generated code !
        //}AFX_MSG

        DECLARE_MESSAGE_MAP()
};

// BrowseDlg.h : header file
//

#ifdef _AFX_BROWSEDLG_H__75414AD6_C1BB_471D_87F8_88768609CC15__INCLUDE
D_)
#define
AFX_BROWSEDLG_H__75414AD6_C1BB_471D_87F8_88768609CC15__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////
/////
// CBrowseDlg dialog

class CBrowseDlg : public CDialog
{
    // Construction
public:
    CBrowseDlg();

    // Dialog Data
    //{AFX_DATA(CBrowseDlg)
    enum { IDD = IDD_BROWSEDLG };
    CListBox    m_lstLabels;

```

```

        int            m_iLabel;
        ///}}AFX_DATA

        CString m_strLabel;
        CString m_strIPAddress;

    // Overrides
        // ClassWizard generated virtual function overrides
        ///{{AFX_VIRTUAL(CBrowseDlg)
        protected:
            virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV support
        ///}}AFX_VIRTUAL

    // Implementation
    protected:

        // Generated message map functions
        ///{{AFX_MSG(CBrowseDlg)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        afx_msg void lstLabels_OnDblClick();
        ///}}AFX_MSG
        DECLARE_MESSAGE_MAP()

    private:
        _bstr_t m_bstrXML;

        bool camlPopulateListWithRowset(_bstr_t bstrXML, CListBox&
lstPopulate);
    };

    ///{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ will insert additional declarations immediately
    before the previous line.

    #endif //
    #ifndef AFX_BROWSEDLG_H__75414AD6_C1BB_471D_87F8_88768609CC15__INCLUDE
    D_
    #endif

    // LookupDlg.cpp : implementation file
    //

    #include "stdafx.h"

    #ifdef _DEBUG
    #define new DEBUG_NEW
    #undef THIS_FILE
    static char THIS_FILE[] = __FILE__;
    #endif

```

```

////////////////////////////////////
////////

```

```

// CLookupDlg dialog

```

```

CLookupDlg::CLookupDlg()
    : CDialog(CLookupDlg::IDD, 0)
{
   //{{AFX_DATA_INIT(CLookupDlg)
    m_strLabel = _T("");
   //}}AFX_DATA_INIT
}

```

```

void CLookupDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CLookupDlg)
    DDX_Control(pDX, IDC_TXT_LABEL, m_txtLabel);
    DDX_Text(pDX, IDC_TXT_LABEL, m_strLabel);
    DDV_MaxChars(pDX, m_strLabel, 50);
   //}}AFX_DATA_MAP
}

```

```

BEGIN_MESSAGE_MAP(CLookupDlg, CDialog)
   //{{AFX_MSG_MAP(CLookupDlg)

        ON_BN_CLICKED(IDC_BTN_BROWSE, btnBrowse_OnClick)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
////////

```

```

// CLookupDlg message handlers

```

```

BOOL CLookupDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

```

```

void CLookupDlg::btnBrowse_OnClick()
{
    CBrowseDlg dlgBrowse;
    if (dlgBrowse.DoModal() == IDCANCEL) return;

    m_strLabel        = dlgBrowse.m_strLabel;
}

```

```

        m_strIPAddress = dlgBrowse.m_strIPAddress;

        EndDialog(IDOK);
    }

void CLookupDlg::OnOK()
{
    CString strDisplay;
    _bstr_t bstrXML;
    int iRowCount;

    UpdateData();

    // Validate the data entry: Label must be at least 4 chars

    if (m_strLabel.GetLength() < 4)
    {
        MessageBox("The Label must be at least 4 characters in
length.", "Video Streaming System", MB_ICONINFORMATION |
MB_SETFOREGROUND);
        m_txtLabel.SetFocus();
        return;
    }

    // Execute the "Display" method against the web server to
lookup the Label

    strDisplay.Format(IDS_CAML_DISPLAY_FILTERED, m_strLabel);
    bstrXML = CVideoPeer::ExecuteWebMethod(strDisplay);

    if (!CVideoPeer::WebMethodSucceeded("Lookup", bstrXML,
CAML_ENDOFROWSET))
    {
        MessageBox("An error occurred while attempting to contact
the video directory server.", "Video Streaming System",
MB_ICONINFORMATION | MB_SETFOREGROUND);
        return;
    }

    // Get the corresponding IP Address

    m_strIPAddress = "ows_IPAddress"; // To get the IP Address, we
must provide the field name
    iRowCount = CVideoPeer::camlQueryRowset(bstrXML, m_strIPAddress);
    if (iRowCount != 1)
    {
        m_strIPAddress.Empty();
        MessageBox("There is no video directory entry matching the
label you entered.\r\nPlease try again or click the Browse button.",
"Video Streaming System", MB_ICONINFORMATION | MB_SETFOREGROUND);
        return;
    }

    // Close the dialog. The caller can obtain the IP Address
from the
    // public member variable m_strIPAddress

```

```

        EndDialog(IDOK);
    }

// LookupDlg.h : header file
//

#ifndef AFX_LOOKUPDLG_H__9425B091_C2DD_4696_ACDF_40CBEE3AFEC7__INCLUDE
D_
#define AFX_LOOKUPDLG_H__9425B091_C2DD_4696_ACDF_40CBEE3AFEC7__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//////////////////////
/////
// CLookupDlg dialog

class CLookupDlg : public CDialog
{
    // Construction
public:
    CLookupDlg();

    // Dialog Data
    //{AFX_DATA(CLookupDlg)
    enum { IDD = IDD_LOOKUPDLG };
    CEdit m_txtLabel;
    CString m_strLabel;
    //}AFX_DATA

    CString m_strIPAddress;           // On success, contains the
IP Address of the entry

    // Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CLookupDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV support
    //}AFX_VIRTUAL

    // Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CLookupDlg)
    virtual BOOL OnInitDialog();
    afx_msg void btnBrowse_OnClick();
    virtual void OnOK();

```

```

        ///}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_LOOKUPDLG_H__9425B091_C2DD_4696_ACDF_40CBEE3AFEC7__INCLUDE
D_

// RegistrationDlg.cpp : implementation file
//

#include "stdafx.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
/////
// CRegistrationDlg dialog

CRegistrationDlg::CRegistrationDlg(LPCTSTR pszCurrentIPAddress)
: CDialog(CRegistrationDlg::IDD, 0)
{
    ///{{AFX_DATA_INIT(CRegistrationDlg)
    m_strLabel = _T("");
    m_strPassword = _T("");
    m_strPassword2 = _T("");
    m_strCurrentIPAddress = pszCurrentIPAddress;
    m_bVisible = TRUE;
    ///}}AFX_DATA_INIT
}

void CRegistrationDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CRegistrationDlg)
    DDX_Control(pDX, IDC_TXT_PASSWORD2, m_txtPassword2);
    DDX_Control(pDX, IDC_TXT_PASSWORD, m_txtPassword);
    DDX_Control(pDX, IDC_TXT_LABEL, m_txtLabel);
    DDX_Text(pDX, IDC_TXT_LABEL, m_strLabel);
    DDV_MaxChars(pDX, m_strLabel, 50);
    DDX_Text(pDX, IDC_TXT_PASSWORD, m_strPassword);

```

```

        DDV_MaxChars(pDX, m_strPassword, 8);
        DDX_Text(pDX, IDC_TXT_PASSWORD2, m_strPassword2);
        DDV_MaxChars(pDX, m_strPassword2, 8);
        DDX_Text(pDX, IDC_TXT_CURRENT_IP, m_strCurrentIPAddress);
        DDV_MaxChars(pDX, m_strCurrentIPAddress, 20);
        DDX_Check(pDX, IDC_CHK_VISIBLE, m_bVisible);
    ///}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CRegistrationDlg, CDialog)
    ///{{AFX_MSG_MAP(CRegistrationDlg)
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CRegistrationDlg message handlers

BOOL CRegistrationDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCK Property Pages should return
FALSE
}

void CRegistrationDlg::OnOK()
{
    UpdateData();

    // Validate the data entry: Label, Password, and Password2
must be
    // at least 4 chars, and Password must equal Password2

    if (m_strLabel.GetLength() < 4)
    {
        MessageBox("The Label must be at least 4 characters in
length.", "Video Streaming System", MB_ICONINFORMATION |
MB_SETFOREGROUND);
        m_txtLabel.SetFocus();
        return;
    }

    if ((m_strPassword.GetLength() < 4) ||
(m_strPassword2.GetLength() < 4))
    {
        MessageBox("The Password must be at least 4 characters in
length.", "Video Streaming System", MB_ICONINFORMATION |
MB_SETFOREGROUND);
        m_txtPassword.SetFocus();
    }
}

```

```

        return;
    }

    if (m_strPassword != m_strPassword2)
    {
        MessageBox("The Password and Password Confirmation fields
do not match, please try again.", "Video Streaming System",
MB_ICONINFORMATION | MB_SETFOREGROUND);
        m_txtPassword.SetFocus();
        return;
    }

    EndDialog(IDOK);
}

// RegistrationDlg.h : header file
//

#ifndef __AFX_REGISTRATIONDLG_H__6DDB6816_817D_4E37_8DC1_5BE3123F2B1F__I
NCLUDED_
#define __AFX_REGISTRATIONDLG_H__6DDB6816_817D_4E37_8DC1_5BE3123F2B1F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////
/////
// CRegistrationDlg dialog

class CRegistrationDlg : public CDialog
{
    // Construction
public:
    CRegistrationDlg(LPCTSTR pszCurrentIPAddress);

    // Dialog Data
    //{AFX_DATA(CRegistrationDlg)
    enum { IDD = IDD_REGISTRATIONDLG };
    CEdit m_txtPassword2;
    CEdit m_txtPassword;
    CEdit m_txtLabel;
    CString      m_strLabel;
    CString      m_strPassword;
    CString      m_strPassword2;
    CString      m_strCurrentIPAddress;
    BOOL m_bVisible;
    //}AFX_DATA

    // Overrides

```

```

        // ClassWizard generated virtual function overrides
        //{AFX_VIRTUAL(CRegistrationDlg)
        protected:
            virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV support
        //}}AFX_VIRTUAL

    // Implementation
    protected:

        // Generated message map functions
        //{AFX_MSG(CRegistrationDlg)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_REGISTRATIONDLG_H__6DDB6816_817D_4E37_8DC1_5BE3123F2B1F__I
NCLUDED_

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by VPD.rc
//
#define ID_BTN_BROWSE                3
#define IDS_PROJNAME                 100
#define IDB_VIDEOPEER                102
#define IDR_VIDEOPEER                103
#define IDS_CAML_SAVE                103
#define IDS_CAML_FIELD_PREFIX_ESCAPED 104
#define IDS_CAML_DISPLAY_FILTERED    105
#define IDS_CAML_DELETE              106
#define IDS_CAML_RESULT_DELETE_CONFIRM 107
#define IDS_CAML_DISPLAY              108
#define IDD_REGISTRATIONDLG          201
#define IDC_LBL_INSTRUCTIONS          201
#define IDC_LBL_LABEL                 202
#define IDD_LOOKUPDLG                202
#define IDC_LBL_PASSWORD              203
#define IDD_BROWSEDLG                203
#define IDC_LBL_CURRENT_IP            204
#define IDC_CHK_VISIBLE               205
#define IDC_LBL_PASSWORD2             206
#define IDC_TXT_LABEL                 207
#define IDC_TXT_PASSWORD              208
#define IDC_TXT_PASSWORD2            209
#define IDC_TXT_CURRENT_IP            210
#define IDC_LBL_PUBLISH_HELP         211
#define IDC_LBL_HELP                  214

```

```

#define IDC_LST_LABELS                215
#define IDC_LBL_SELECT                216

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        204
#define _APS_NEXT_COMMAND_VALUE        32768
#define _APS_NEXT_CONTROL_VALUE        217
#define _APS_NEXT_SYMED_VALUE          104
#endif
#endif
// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>

#include <statreg.cpp>
#endif

#include <atlimpl.cpp>

HRESULT ResultMsgBox
(
    HRESULT hr,
    LPCTSTR szAppMsg
)
/*
    DESCRIPTION
        Display a MessageBox with the description of the error indicated
    by
        the passed-in HRESULT value, which should be a System-
    defined
        Result Code.

        Return the passed-in Result Code
*/
{
    LPTSTR szSysMsg;

    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM,
        NULL,
        hr,
        MAKELANGID (LANG_NEUTRAL,
        SUBLANG_DEFAULT),
        (LPTSTR) &szSysMsg,
        0,
        NULL);

```

```

        TCHAR szMsgBuf[300];
        if (szSysMsg) wsprintf(szMsgBuf, "%s\n\n%s(%lx)", szAppMsg,
szSysMsg, hr);
        else wsprintf(szMsgBuf, "%s\n\n(Error code 0x%lx)", szAppMsg,
hr);

        MessageBox(NULL, szMsgBuf, "Video Streaming System",
MB_OK|MB_ICONWARNING|MB_SETFOREGROUND); //|MB_SYSTEMMODAL);

        LocalFree(szSysMsg);

        return S_OK;
//    return hr;
}

void DoEvents()
{
    MSG msg;

    while (::PeekMessage(&msg, NULL, NULL, NULL, PM_NOREMOVE))
    {
        AfxGetThread()->PumpMessage();
    }
}

// stdafx.h : include file for standard system include files,
//           or project specific include files that are used frequently,
//           but are changed infrequently

#ifdef _MSC_VER
#pragma once
#endif

#define STRICT
#ifdef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
#define _ATL_APARTMENT_THREADED

#include <afxwin.h>
#include <afxdisp.h>

#include <atlbase.h>

#include <stdexcept>
#include <winsock2.h>

#import "msxml4.dll"

```

```

//You may derive a class from CComModule and use it if you want to
override
//something, but do not change the name of _Module
extern CComModule _Module;

#include <atlcom.h>
#include <atlctl.h>

#include "resource.h"
#include "VPD.h"
#include "VPDApp.h"
#include "RegistrationDlg.h"
#include "BrowseDlg.h"
#include "LookupDlg.h"
#include "VideoPeer.h"

extern class CVPDApp theApp;

// Global fxn declarations
HRESULT ResultMsgBox(HRESULT hr, LPCTSTR szAppMsg);
void DoEvents();

// #include "C:\Documents and Settings\slathrop\Desktop\DCOMHelp.h"

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__29BE88B1_E6DA_431D_8266_7B4B31AED45C__INCLUDED)

// VideoPeer.cpp : Implementation of CVideoPeer

#include "stdafx.h"

#define YP_DEFAULT_BASE_URL
"http://na1ay.sharepoint.bcentral.com/bklk/_vti_bin/owssvr.dll?"

////////////////////////////////////
/////
// CVideoPeer

CString CVideoPeer::m_strBaseUrl = YP_DEFAULT_BASE_URL;

CVideoPeer::CVideoPeer()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())

```

```

        m_strBaseURL = theApp.GetProfileString("Y llowPages", "BaseURL",
YP_DEFAULT_BASE_URL);
    }

STDMETHODIMP CVideoPeer::Register
(
    IN BSTR                Label,
    IN BSTR                Password,
    IN VARIANT_BOOL Visible,
    IN VARIANT_BOOL Prompt,
    IN BSTR                IPAddress
)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    CString strLabel, strPassword, strVisible, strIPAddress;
    CString strDisplay, strSave, strUpdateURL;
    CString strCurrentIPAddress(GetCurrentIPAddress()); // We get
the IP Address here
    _bstr_t bstrXML;
    CString strPwdOnServer, strID;
    int      iRowCount;

    // Fail if we couldn't determine the current IP Address

    if (strCurrentIPAddress.IsEmpty())
        return ResultMsgBox(E_FAIL, "Unable to register this Video
Peer. The current IP Address could not be determined.");

    // Get any cached information for this peer

    strLabel      = theApp.GetProfileString("ThisPeer", "Label");
    strPassword    = theApp.GetProfileString("ThisPeer", "Password");
    strVisible     = theApp.GetProfileString("ThisPeer", "Visible", "-
1");
    strIPAddress  = theApp.GetProfileString("ThisPeer", "IPAddress",
strCurrentIPAddress);

    // If the label or password wasn't cached, we must prompt
the user

    if (strLabel.IsEmpty() || strPassword.IsEmpty())
    {
        CRegistrationDlg dlgRegister(strCurrentIPAddress);
        if (dlgRegister.DoModal() == IDCANCEL)
        {
            goto Exit1;
            return E_ABORT;
        }

        // Prepare the user-entered data for registration

        strLabel      = dlgRegister.m_strLabel;
        strPassword    = dlgRegister.m_strPassword;
        strVisible     = dlgRegister.m_bVisible ? "-1" : "0";
        strIPAddress  = strCurrentIPAddress;
    }
}

```

```

    }
    else
    {
        // The label and password *are* cached, so
Registration    // is unnecessary if the IP Address hasn't changed

        if (strIPAddress == strCurrentIPAddress) return S_OK;

        // The current IP Address has changed, so prepare to
register it

        strIPAddress = strCurrentIPAddress;
    }

    // The IP Address has changed, or this is a new
registration, so...
    // Execute the "Display" method against the web server to
lookup
    // the Label, verifying that the Label either:
    // (a) Doesn't already exist, or
    // (b) Exists already, but the password we supplied matches
the one on the server

    strDisplay.Format(IDS_CAML_DISPLAY_FILTERED, strLabel);
    bstrXML = ExecuteWebMethod(strDisplay);

    if (!WebMethodSucceeded("Lookup", bstrXML, CAML_ENDOFROWSET))
        return ResultMsgBox(E_FAIL, "An error occurred while
attempting to contact the video directory server.");

    strPwdOnServer = "ows_Password";    // To get the current
password on the server, we must provide the field name
    iRowCount = camlQueryRowset(bstrXML, strPwdOnServer);
    if (iRowCount != 0)
    {
        if ((iRowCount > 1) || (strPwdOnServer == "ows_Password"))
return ResultMsgBox(E_FAIL, "The video directory data for the label you
entered is corrupted. Please choose another label.");
        if (strPassword != strPwdOnServer) return
ResultMsgBox(E_FAIL, "The video directory label you entered is already
in use, and the password you supplied does not match the one on file.
Please choose another label or provide the correct password.");

        // The Label already exists and the password is
correct, so perform an "Update"...

        // First get the ID of the entry

        strID = "ows_ID";
        iRowCount = camlQueryRowset(bstrXML, strID);
        if (iRowCount == -1) return ResultMsgBox(E_FAIL, "The video
directory data for the label you entered is corrupted. Please choose
another label.");
    }

```

```

//NOTE: Th  SPTS "batch" command format doesn't work
properly when
//          performing an Update. The URL command
format must be used instead.
//          Notice that we have created a custom
XML response file on the
//          SPTS server (URLCmdConfirm.xml), which
is returned on success
//          because we included the "NextUsing" arg
in the URL.

```

```

// Now build the URL-formatted SPTS web method for
updating the entry

```

```

    strUpdateURL = m_strBaseURL +
    "Cmd=Save&List=u_VPD&NextUsing=URLCmdConfirm.xml&ID=" + strID;
    strUpdateURL += "&" + CString(LPCTSTR)
IDS_CAML_FIELD_PREFIX_ESCAPED) + "IPAddress=";
    strCurrentIPAddress.Replace(".", "%2E");
    strUpdateURL += strCurrentIPAddress;

    // Execute the web method

    bstrXML = ExecuteWebMethod(0, strUpdateURL);
    if (!WebMethodSucceeded("URLCmd", bstrXML)) return
ResultMsgBox(E_FAIL, "An error occurred while attempting to update the
entry on the video directory server.");
}
else
{

```

```

    // Execute the "Save" method against the web server

    strSave.Format(IDS_CAML_SAVE, strLabel, strPassword,
strVisible, strIPAddress);
    bstrXML = ExecuteWebMethod(strSave);
    if (!WebMethodSucceeded("Register", bstrXML)) return
ResultMsgBox(E_FAIL, "An error occurred while attempting to save the
entry to the video directory server.");
}

```

```

    theApp.WriteProfileString("ThisPeer", "Label", strLabel);
    theApp.WriteProfileString("ThisPeer", "Password", strPassword);
    theApp.WriteProfileString("ThisPeer", "Visible", strVisible);
    theApp.WriteProfileString("ThisPeer", "IPAddress", strIPAddress);
Exit1:
    return S_OK;
}

```

```

STDMETHODIMP CVideoPeer::Unregister
(
    IN BSTR          Label,
    IN BSTR          Password,
    IN VARIANT_BOOL Prompt
)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
}

```

```

CString strLabel, strPassword;
CString strDisplay, strDelete;
_bstr_t bstrXML;
CString strPwdOnServer, strID;
int      iRowCount;

    // Get any cached information for this peer

strLabel    = theApp.GetProfileString("ThisPeer", "Label");
strPassword = theApp.GetProfileString("ThisPeer", "Password");

    // If the label or password wasn't cached, we have nothing
to do      // (for now, we ignore args provided to this method)

    if (strLabel.IsEmpty() || strPassword.IsEmpty()) return S_OK;

    // Execute the "Display" method against the web server to
lookup     // the Label, verifying that the Label either:
           // (a) Doesn't exist, or
           // (b) Exists, and the password we have matches the one on
the server

    strDisplay.Format(IDS_CAML_DISPLAY_FILTERED, strLabel);
    bstrXML = ExecuteWebMethod(strDisplay);

    if (!WebMethodSucceeded("Lookup", bstrXML, CAML_ENDOFROWSET))
        return ResultMsgBox(E_FAIL, "An error occurred while
attempting to contact the video directory server.");

    strPwdOnServer = "ows_Password"; // To get the current
password on the server, we must provide the field name
    iRowCount = camlQueryRowset(bstrXML, strPwdOnServer);
    if (iRowCount == 0) goto end;

    if ((iRowCount > 1) || (strPwdOnServer == "ows_Password")) return
ResultMsgBox(E_FAIL, "The video directory data for your computer is
corrupted. Please choose another label.");
    if (strPassword != strPwdOnServer) return ResultMsgBox(E_FAIL,
"The video directory label you entered was found on the server, but the
password you supplied does not match the one on file. Please choose
another label or provide the correct password.");

    strID = "ows_ID"; // To get the ID, we must provide the field
name
    iRowCount = camlQueryRowset(bstrXML, strID);

    // Execute the "Delete" method against the web server

    //NOTE: The SPTS "batch" command format doesn't work properly
when      //
           // performing a Delete. The URL command format must be
used instead.
           // Notice that we have created a custom XML response
file on the

```

```

//          SPTS server (URLCmdConfirm.xml), which is returned
on success
//          because we included the "NextUsing" arg in the URL.
//
//strDelete.Format(IDS_CAML_DELETE, strID);
//bstrXML = ExecuteWebMethod(strDelete);

bstrXML = ExecuteWebMethod(0, m_strBaseURL +
"Cmd=Delete&List=u_VPD&NextUsing=URLCmdConfirm.xml&ID=" + strID);
if (!WebMethodSucceeded("URLCmd", bstrXML)) return
ResultMsgBox(E_FAIL, "An error occurred while attempting to remove the
entry from the video directory server.");

```

end:

```

// Delete the local registry entries

theApp.WriteProfileString("ThisPeer", "Label", 0);
theApp.WriteProfileString("ThisPeer", "Password", 0);
theApp.WriteProfileString("ThisPeer", "Visible", 0);
theApp.WriteProfileString("ThisPeer", "IPAddress", 0);

return S_OK;
}

STDMETHODIMP CVideoPeer::Lookup
(
    IN          LookupPromptType Prompt,          // Default =
lptList
    IN OUT BSTR*          Label,
    OUT      BSTR*          IPAddress
)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    USES_CONVERSION;

    // Validate args

    if (IPAddress == NULL) return E_POINTER;
    if (Prompt == lptNone)
    {
        if (Label == NULL) return E_POINTER;
        if (SysStringLen(*Label) < 4) return E_INVALIDARG;
    }

    // Proceed with lookup according to type of Prompt
specified

    if (Prompt == lptList)
    {
        // Show "Browse" dialog with list of labels

        CBrowseDlg dlgBrowse;
        if (dlgBrowse.DoModal() == IDCANCEL) return E_ABORT;
    }
}

```

```

        // Set the output args

        *Label      = dlgBrowse.m_strLabel.AllocSysString();
        *IPAddress = dlgBrowse.m_strIPAddress.AllocSysString();
    }
    else if (Prompt == lptInput)
    {
        // Show Lookup dlg, allowing user to input a label or
click "Browse"
        // for a list

        CLookupDlg dlgLookup;
        if (dlgLookup.DoModal() == IDCANCEL) return E_ABORT;

        // Set the output args

        if (!dlgLookup.m_strIPAddress.IsEmpty())
        {
            *Label      =
dlgLookup.m_strLabel.AllocSysString();
            *IPAddress =
dlgLookup.m_strIPAddress.AllocSysString();
        }
        else if (Prompt == lptNone)
        {
            CString strDisplay, strIPAddress;
            _bstr_t bstrXML;
            int      iRowCount;

            // Execute the "Display" method against the web
server to lookup the Label

            strDisplay.Format(IDS_CAML_DISPLAY_FILTERED,
OLE2CT(*Label));
            bstrXML = ExecuteWebMethod(strDisplay);

            if (!WebMethodSucceeded("Lookup", bstrXML,
CAML_ENDOFROWSET)) return RPC_E_FAULT;

            // Get the corresponding IP Address

            strIPAddress = "ows_IPAddress";    // To get the IP
Address, we must provide the field name
            iRowCount = camlQueryRowset(bstrXML, strIPAddress);
            if (iRowCount != 1) return INET_E_OBJECT_NOT_FOUND;    // No
matching label in "Yellow Pages" Directory
            *IPAddress = strIPAddress.AllocSysString();
        }
        else return E_INVALIDARG;

        return S_OK;
    }

    // Execute a SPTS "Web Method" and return the result as a string.
    // If successful, the returned string is an XML document that

```

```

// contains a success code. If unsuccessful, the string is
// probably an HTML document with a SPTS error message
//
_bstr_t CVideoPeer::ExecuteWebMethod
(
    LPCTSTR pszBody,                // default = NULL
    LPCTSTR pszMethodURL,           // default = m_strBaseUrl +
    "Cmd=DisplayPost"
    LPCTSTR pszVerb,                // default = "POST"
    LPCTSTR pszCustomHeaderName,    // default = "Content-type"
    LPCTSTR pszCustomHeaderValue   // default = "application/x-www-
form-urlencoded"
)
{
    CWaitCursor wait;
    IXMLHTTPRequestPtr spRequest;

    try
    {
        // Send the request and get the response

        spRequest.CreateInstance("Msxml2.XMLHTTP.4.0");

        spRequest->open(pszVerb, pszMethodURL, false);
        spRequest->setRequestHeader(pszCustomHeaderName,
pszCustomHeaderValue);
        spRequest->send(_variant_t(pszBody));

        return spRequest->responseText;
    }
    catch (...)
    {
        return _bstr_t();
    }
}

// Return true if the CAML result code matches the one we expected,
// return false otherwise
//
bool CVideoPeer::WebMethodSucceeded
(
    LPCTSTR pszMethodName,
    _bstr_t bstrMethodResponseXML,
    _bstr_t bstrSuccessCode          // The success code as
a BSTR, default = "0"
)
{
    CString                strXPath;
    IXMLDOMDocument2Ptr spXML;
    IXMLDOMNodePtr        spMethodResultNode;
    VARIANT_BOOL           vntbRes;

    try
    {
        // Prepare the XPath query for the result code

```

```

        strXPath.Format("//Result[@ID = '%s']", pszMethodName);

        // Parse the XML response for the method's result
code

        spXML.CreateInstance("Msxml2.DOMDocument.4.0");

        vntbRes = spXML->loadXML(bstrMethodResponseXML);
        if (vntbRes == VARIANT_FALSE) throw exception();
        spXML->setProperty("SelectionLanguage", "XPath");
        spMethodResultNode = spXML->selectSingleNode(_bstr_t(strXPath));

        // If the "Code" attribute matches the expected
        success code, the call was successful

        return (spMethodResultNode->attributes->
>getNamedItem("Code")->text == bstrSuccessCode);
    }
    catch (...)
    {
        return false;
    }
}

// If a field name (2nd arg) is supplied, set the field string to the
// value of
// that field found in the first row (or the row specified by
// iRowIndex) of the
// result set, if any. If iRowIndex is greater than the number of rows
// in the
// XML string, return the value in the last row.
// Set strField to "" on failure.
// Return -1 on failure. On success, return the count of data rows
// found in the XML string
//
int CVideoPeer::camlQueryRowset
(
    _bstr_t      bstrXML,
    CString& strField,
    int          iRowIndex // Default = 0 (first row)
)
{
    IXMLDOMDocument2Ptr spXML;
    IXMLDOMNodeListPtr spNodeList;
    IXMLDOMNodePtr      spNode;
    VARIANT_BOOL         vntbRes;
    int                  iRowCount, iRow=0;

    try
    {
        // Parse the XML response for the CAML data rows

        spXML.CreateInstance("Msxml2.DOMDocument.4.0");

```

```

vntbRes = spXML->loadXML(bstrXML);
if (vntbRes == VARIANT_FALSE) throw exception();
spXML->setProperty("SelectionLanguage", "XPath");
spXML->setProperty("SelectionNamespaces",
"xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882' "
"xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882' "
"xmlns:rs='urn:schemas-microsoft-com:rowset' "
"xmlns:z='#RowsetSchema'");
spNodeList = spXML->selectNodes("//z:row");
iRowCount = spNodeList->length;
if (iRowCount < 1)
{
    strField.Empty();
    return iRowCount;
}

for (spNode=spNodeList->nextNode(); (spNode != NULL) &&
(iRow < iRowIndex); spNode=spNodeList->nextNode(), iRow++);

    strField = (BSTR) spNode->attributes-
>getNamedItem(_bstr_t(strField))->text;

    return iRowCount;
}
catch (...)
{
    strField = "";
    return -1;
}
}

CString CVideoPeer::GetCurrentIPAddress()
{
    CString    strIPAddress;
    WSADATA    wsad;
    int        err;
    char        szHostName[1024];
    HOSTENT*    pHE;
    IN_ADDR     inaddr;

    err = WSStartup(MAKEWORD(1,1), &wsad);
    if (err) return CString();

    err = gethostname(szHostName, 1024);

    if (err) goto end;

    pHE = gethostbyname(szHostName);
    if (!pHE) goto end;

```

```

        memcpy(&inaddr, pHE->h_addr, 4);
        strIPAddress = inet_ntoa(inaddr);

end:

        WSACleanup();
        return strIPAddress;
}

// VideoPeer.h : Declaration of the CVideoPeer

#ifndef __VIDEOPEER_H_
#define __VIDEOPEER_H_

#include "resource.h"           // main symbols
#include <atlctl.h>

#include <afxinet.h>

using namespace MSXML2;

#define HTTP_STATUS_MULTISTATUS      207
#define HTTP_READ_BUFFER_SIZE       4096

#define CAML_ENDOFROWSET              "265926"

////////////////////////////////////
/////
// CVideoPeer
class ATL_NO_VTABLE CVideoPeer :
    public CComObjectRootEx<CComSingleThreadModel>,
    public IDispatchImpl<IVideoPeer, &IID_IVideoPeer,
&LIBID_VPDModule>,
    public CComControl<CVideoPeer>,
    public IPersistStreamInitImpl<CVideoPeer>,
    public IOleControlImpl<CVideoPeer>,
    public IOleObjectImpl<CVideoPeer>,
    public IOleInPlaceActiveObjectImpl<CVideoPeer>,
    public IViewObjectExImpl<CVideoPeer>,
    public IOleInPlaceObjectWindowlessImpl<CVideoPeer>,
    public ISupportErrorInfo,
    public IPersistStorageImpl<CVideoPeer>,
    public ISpecifyPropertyPagesImpl<CVideoPeer>,
    public IQuickActivateImpl<CVideoPeer>,
    public IDataObjectImpl<CVideoPeer>,
    public IProvideClassInfo2Impl<&CLSID_VideoPeer, NULL,
&LIBID_VPDModule>,
    public CComCoClass<CVideoPeer, &CLSID_VideoPeer>
{
    public:

        CVideoPeer();

```

```

DECLARE_REGISTRY_RESOURCEID(IDR_VIDEOPEER)
DECLARE_NOT_AGGREGATABLE(CVideoPeer)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CVideoPeer)
    COM_INTERFACE_ENTRY(IVideoPeer)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(IViewObjectEx)
    COM_INTERFACE_ENTRY(IViewObject2)
    COM_INTERFACE_ENTRY(IViewObject)
    COM_INTERFACE_ENTRY(IoleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IoleInPlaceObject)
    COM_INTERFACE_ENTRY2(IoleWindow,
IoleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IoleInPlaceActiveObject)
    COM_INTERFACE_ENTRY(IoleControl)
    COM_INTERFACE_ENTRY(IoleObject)
    COM_INTERFACE_ENTRY(IPersistStreamInit)
    COM_INTERFACE_ENTRY2(IPersist, IPersistStreamInit)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY(ISpecifyPropertyPages)
    COM_INTERFACE_ENTRY(IQuickActivate)
    COM_INTERFACE_ENTRY(IPersistStorage)
    COM_INTERFACE_ENTRY(IDataObject)
    COM_INTERFACE_ENTRY(IProvideClassInfo)
    COM_INTERFACE_ENTRY(IProvideClassInfo2)
END_COM_MAP()

BEGIN_PROP_MAP(CVideoPeer)
    PROP_DATA_ENTRY("_cx", m_sizeExtent.cx, VT_UI4)
    PROP_DATA_ENTRY("_cy", m_sizeExtent.cy, VT_UI4)
    // Example entries
    // PROP_ENTRY("Property Description", dispid, clsid)
    // PROP_PAGE(CLSID_StockColorPage)
END_PROP_MAP()

BEGIN_MSG_MAP(CVideoPeer)
    CHAIN_MSG_MAP(CComControl<CVideoPeer>)
    DEFAULT_REFLECTION_HANDLER()
END_MSG_MAP()
// Handler prototypes:
// LRESULT MessageHandler(UINT uMsg, WPARAM wParam, LPARAM
lParam, BOOL& bHandled);
// LRESULT CommandHandler(WORD wNotifyCode, WORD wID, HWND
hWndCtl, BOOL& bHandled);
// LRESULT NotifyHandler(int idCtrl, LPNMHDR pnmh, BOOL&
bHandled);

HRESULT OnDraw(ATL_DRAWINFO& di)
{
    RECT& rc = *(RECT*)&di.prcBounds;
    Rectangle(di.hdcDraw, rc.left, rc.top, rc.right,
rc.bottom);

    SetTextAlign(di.hdcDraw, TA_CENTER|TA_BASELINE);

```

```

        LPCTSTR pszText = _T("Video Peer Directory OCX");
        TextOut(di.hdcDraw,
            (rc.left + rc.right) / 2,
            (rc.top + rc.bottom) / 2,
            pszText,
            lstrlen(pszText));

        return S_OK;
    }

// ISupportsErrorInfo

STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_IVideoPeer,
    };
    for (int i=0; i<sizeof(arr)/sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i], riid))
            return S_OK;
    }
    return S_FALSE;
}

// IViewObjectEx

DECLARE_VIEW_STATUS(VIEWSTATUS_SOLIDBKGND |
VIEWSTATUS_OPAQUE)

// IVideoPeer

public:

    STDMETHOD(Register)(IN BSTR Label, IN BSTR Password, IN
        VARIANT_BOOL Visible, IN VARIANT_BOOL Prompt, IN BSTR IPAddress);
    STDMETHOD(Unregister)(IN BSTR Label, IN BSTR Password, IN
        VARIANT_BOOL Prompt);
    STDMETHOD(Lookup)(IN LookupPromptType Prompt, IN OUT BSTR*
        Label, OUT BSTR* IPAddress);

// CVideoPeer

friend class CLookupDlg;
friend class CBrowseDlg;

private:

    static CString m_strBaseURL;

    // We declare the following methods as "static" so that the
    friend classes

```

```

        // can call them without having a pointer to an instance of
this class

```

```

        static _bstr_t ExecuteWebMethod
            (LPCTSTR pszBody
            = NULL,
            LPCTSTR pszMethodURL
            = m_strBaseURL + "Cmd=DisplayPost",
            LPCTSTR pszVerb
            = "POST",
            LPCTSTR pszCustomHeaderName = "Content-
type",
            LPCTSTR pszCustomHeaderValue =
"application/x-www-form-urlencoded");
        static bool WebMethodSucceeded
            (LPCTSTR pszMethodName,
            _bstr_t bstrMethodResponseXML,
            _bstr_t bstrSuccessCode = "0");
        static int camlQueryRowset(_bstr_t bstrXML, CString&
strField = CString(""), int iRowIndex = 0);
        static CString GetCurrentIPAddress();
};

```

```

#endif // __VIDEOPEER_H_

```

```

// VPD.cpp : Implementation of DLL Exports.

```

```

// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f VPDps.mk in the project directory.

```

```

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "VPD.h"
#include "VPD_i.c"

```

```

CComModule _Module;

```

```

////////////////////////////////////
/////

```

```

// Used to determine whether the DLL can be unloaded by OLE

```

```

STDAPI DllCanUnloadNow(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return (AfxDllCanUnloadNow()==S_OK && _Module.GetLockCount()==0) ?
S_OK : S_FALSE;
}

```

```

/////////////////////////////////////////////////////////////////
/////
// Returns a class factory to creat  an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    return _Module.GetClassObject(rclsid, riid, ppv);
}

/////////////////////////////////////////////////////////////////
/////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
    // registers object, typelib and all interfaces in typelib
    return _Module.RegisterServer(TRUE);
}

/////////////////////////////////////////////////////////////////
/////
// DllUnregisterServer - Removes entries from the system registry

STDAPI DllUnregisterServer(void)
{
    return _Module.UnregisterServer(TRUE);
}

/* this ALWAYS GENERATED file contains the definitions for the
interfaces */

/* File created by MIDL compiler version 5.01.0164 */
/* at Tue Jan 22 00:00:57 2002
*/
/* Compiler settings for D:\CLIENTS\BKLK\VPD\OCX\VPD.idl:
    Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
    error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )

/* verify that the <rpcndr.h> version is high enough to compile this
file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __VPD_h__
#define __VPD_h__

#ifdef __cplusplus
extern "C" {

```

```

#endif

/* Forward Declarations */

#ifndef __IVideoPeer_FWD_DEFINED__
#define __IVideoPeer_FWD_DEFINED__
typedef interface IVideoPeer IVideoPeer;
#endif /* __IVideoPeer_FWD_DEFINED__ */

#ifndef __VideoPeer_FWD_DEFINED__
#define __VideoPeer_FWD_DEFINED__

#ifdef __cplusplus
typedef class VideoPeer VideoPeer;
#else
typedef struct VideoPeer VideoPeer;
#endif /* __cplusplus */

#endif /* __VideoPeer_FWD_DEFINED__ */

/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

#ifndef __VPDModule_LIBRARY_DEFINED__
#define __VPDModule_LIBRARY_DEFINED__

/* library VPDModule */
/* [helpstring][version][uuid] */

typedef /* [helpstring][v1_enum] */
enum LookupPromptType
{
    lptList      = 0x1,
    lptInput     = 0x2,
    lptNone      = 0x3
} LookupPromptType;

EXTERN_C const IID LIBID_VPDModule;

#ifndef __IVideoPeer_INTERFACE_DEFINED__
#define __IVideoPeer_INTERFACE_DEFINED__

/* interface IVideoPeer */
/*
[unique][helpstring][hidden][oleautomation][nonextensible][dual][uuid][
object] */

```

```

EXTERN_C const IID IID_IVideoPeer;

```

```

#if defined(__cplusplus) && !defined(CINTERFACE)

    MIDL_INTERFACE("13B9F9C8-B3AD-445C-9D62-564390A549FB")
    IVideoPeer : public IDispatch
    {
    public:
        virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
Register(
    /* [defaultvalue][in] */ BSTR Label = L"",
    /* [defaultvalue][in] */ BSTR Password = L"",
    /* [defaultvalue][in] */ VARIANT_BOOL Visible = -1,
    /* [defaultvalue][in] */ VARIANT_BOOL Prompt = -1,
    /* [defaultvalue][in] */ BSTR IPAddress = L"") = 0;

        virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
Unregister(
    /* [defaultvalue][in] */ BSTR Label = L"",
    /* [defaultvalue][in] */ BSTR Password = L"",
    /* [defaultvalue][in] */ VARIANT_BOOL Prompt = -1) = 0;

        virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
Lookup(
    /* [defaultvalue][in] */ LookupPromptType Prompt,
    /* [defaultvalue][out][in] */ BSTR __RPC_FAR *Label,
    /* [retval][out] */ BSTR __RPC_FAR *IPAddress) = 0;

    };

#else /* C style interface */

    typedef struct IVideoPeerVtbl
    {
        BEGIN_INTERFACE

        HRESULT ( STDMETHODCALLTYPE __RPC_FAR *QueryInterface )(
            IVideoPeer __RPC_FAR * This,
            /* [in] */ REFIID riid,
            /* [iid_is][out] */ void __RPC_FAR * __RPC_FAR *ppvObject);

        ULONG ( STDMETHODCALLTYPE __RPC_FAR *AddRef )(
            IVideoPeer __RPC_FAR * This);

        ULONG ( STDMETHODCALLTYPE __RPC_FAR *Release )(
            IVideoPeer __RPC_FAR * This);

        HRESULT ( STDMETHODCALLTYPE __RPC_FAR *GetTypeInfoCount )(
            IVideoPeer __RPC_FAR * This,
            /* [out] */ UINT __RPC_FAR *pctinfo);

        HRESULT ( STDMETHODCALLTYPE __RPC_FAR *GetTypeInfo )(
            IVideoPeer __RPC_FAR * This,
            /* [in] */ UINT iTInfo,
            /* [in] */ LCID lcid,
            /* [out] */ ITypeInfo __RPC_FAR * __RPC_FAR *ppTInfo);

        HRESULT ( STDMETHODCALLTYPE __RPC_FAR *GetIDsOfNames )(
            IVideoPeer __RPC_FAR * This,

```

```

        /* [in] */ REFIID riid,
        /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
        /* [in] */ UINT cNames,
        /* [in] */ LCID lcid,
        /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

    /* [local] */ HRESULT ( STDMETHODCALLTYPE __RPC_FAR *Invoke )(
        IVideoPeer __RPC_FAR * This,
        /* [in] */ DISPID dispIdMember,
        /* [in] */ REFIID riid,
        /* [in] */ LCID lcid,
        /* [in] */ WORD wFlags,
        /* [out][in] */ DISPPARAMS __RPC_FAR *pDispParams,
        /* [out] */ VARIANT __RPC_FAR *pVarResult,
        /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
        /* [out] */ UINT __RPC_FAR *puArgErr);

    /* [helpstring][id] */ HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*Register )(
        IVideoPeer __RPC_FAR * This,
        /* [defaultvalue][in] */ BSTR Label,
        /* [defaultvalue][in] */ BSTR Password,
        /* [defaultvalue][in] */ VARIANT_BOOL Visible,
        /* [defaultvalue][in] */ VARIANT_BOOL Prompt,
        /* [defaultvalue][in] */ BSTR IPAddress);

    /* [helpstring][id] */ HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*Unregister )(
        IVideoPeer __RPC_FAR * This,
        /* [defaultvalue][in] */ BSTR Label,
        /* [defaultvalue][in] */ BSTR Password,
        /* [defaultvalue][in] */ VARIANT_BOOL Prompt);

    /* [helpstring][id] */ HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*Lookup )(
        IVideoPeer __RPC_FAR * This,
        /* [defaultvalue][in] */ LookupPromptType Prompt,
        /* [defaultvalue][out][in] */ BSTR __RPC_FAR *Label,
        /* [retval][out] */ BSTR __RPC_FAR *IPAddress);

    END_INTERFACE
} IVideoPeerVtbl;

interface IVideoPeer
{
    CONST_VTBL struct IVideoPeerVtbl __RPC_FAR *lpVtbl;
};

#ifdef COBJMACROS

#define IVideoPeer_QueryInterface(This,riid,ppvObject) \
    (This)->lpVtbl->QueryInterface(This,riid,ppvObject)

#define IVideoPeer_AddRef(This) \

```

```

(This)->lpVtbl -> AddRef(This)

#define IVideoPeer_Release(This) \
    (This)->lpVtbl -> Release(This)

#define IVideoPeer_GetTypeInfoCount(This,pctinfo) \
    (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define IVideoPeer_GetTypeInfo(This, iTInfo, lcid, ppTInfo) \
    (This)->lpVtbl -> GetTypeInfo(This, iTInfo, lcid, ppTInfo)

#define
IVideoPeer_GetIDsOfNames(This, riid, rgszNames, cNames, lcid, rgDispId)
    \
    (This)->lpVtbl ->
GetIDsOfNames(This, riid, rgszNames, cNames, lcid, rgDispId)

#define
IVideoPeer_Invoke(This, dispIdMember, riid, lcid, wFlags, pDispParams, pVarRe
sult, pExcepInfo, puArgErr) \
    (This)->lpVtbl ->
Invoke(This, dispIdMember, riid, lcid, wFlags, pDispParams, pVarResult, pExcep
Info, puArgErr)

#define
IVideoPeer_Register(This, Label, Password, Visible, Prompt, IPAddress) \
    (This)->lpVtbl ->
Register(This, Label, Password, Visible, Prompt, IPAddress)

#define IVideoPeer_Unregister(This, Label, Password, Prompt) \
    (This)->lpVtbl -> Unregister(This, Label, Password, Prompt)

#define IVideoPeer_Lookup(This, Prompt, Label, IPAddress) \
    (This)->lpVtbl -> Lookup(This, Prompt, Label, IPAddress)

#endif /* COBJMACROS */

#endif /* C style interface */

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IVideoPeer_Register_Proxy(
    IVideoPeer __RPC_FAR * This,
    /* [defaultvalue][in] */ BSTR Label,
    /* [defaultvalue][in] */ BSTR Password,
    /* [defaultvalue][in] */ VARIANT_BOOL Visible,
    /* [defaultvalue][in] */ VARIANT_BOOL Prompt,
    /* [defaultvalue][in] */ BSTR IPAddress);

void __RPC_STUB IVideoPeer_Register_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,

```

```

PRPC_MESSAGE _pRpcMessage,
DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IVideoPeer_Unregister_Proxy(
    IVideoPeer __RPC_FAR * This,
    /* [defaultvalue][in] */ BSTR Label,
    /* [defaultvalue][in] */ BSTR Password,
    /* [defaultvalue][in] */ VARIANT_BOOL Prompt);

void __RPC_STUB IVideoPeer_Unregister_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IVideoPeer_Lookup_Proxy(
    IVideoPeer __RPC_FAR * This,
    /* [defaultvalue][in] */ LookupPromptType Prompt,
    /* [defaultvalue][out][in] */ BSTR __RPC_FAR *Label,
    /* [retval][out] */ BSTR __RPC_FAR *IPAddress);

void __RPC_STUB IVideoPeer_Lookup_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

#endif /* __IVideoPeer_INTERFACE_DEFINED__ */

EXTERN_C const CLSID CLSID_VideoPeer;

#ifdef __cplusplus

class DECLSPEC_UUID("434770D6-FB51-4206-96CD-DD7CE811A338")
VideoPeer;
#endif
#endif /* __VPDModule_LIBRARY_DEFINED__ */

/* Additional Prototypes for ALL interfaces */

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif

#endif
// VPDApp.cpp: Implementation of CVPDApp

```



```

CBrowseDlg::CBrowseDlg()
    : CDialog(CBrowseDlg::IDD, 0)
{
    //{AFX_DATA_INIT(CBrowseDlg)
    m_iLabel = -1;
    //}AFX_DATA_INIT
}

void CBrowseDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CBrowseDlg)
    DDX_Control(pDX, IDC_LST_LABELS, m_lstLabels);
    DDX_LBIndex(pDX, IDC_LST_LABELS, m_iLabel);
    //}AFX_DATA_MAP
    DDX_LBString(pDX, IDC_LST_LABELS, m_strLabel);
}

BEGIN_MESSAGE_MAP(CBrowseDlg, CDialog)
    //{AFX_MSG_MAP(CBrowseDlg)
    ON_LBN_DBLCLK(IDC_LST_LABELS, lstLabels_OnDblClick)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
/////
// CBrowseDlg message handlers

BOOL CBrowseDlg::OnInitDialog()
{
    CWaitCursor wait;
    CString strDisplay((LPCTSTR) IDS_CAML_DISPLAY);

    CDialog::OnInitDialog();

    m_bstrXML = CVideoPeer::ExecuteWebMethod(strDisplay);

    if (!CVideoPeer::WebMethodSucceeded("Lookup", m_bstrXML,
CAML_ENDOFROWSET))
    {
        MessageBox("An error occurred while attempting to contact
the video directory server.", "Video Streaming System",
MB_ICONINFORMATION | MB_SETFOREGROUND);
        EndDialog(IDCANCEL);
        return TRUE;
    }

    if (!camlPopulateListWithRowset(m_bstrXML, m_lstLabels))
    {
        MessageBox("An error occurred while attempting to download
the list of video senders.", "Video Streaming System",
MB_ICONINFORMATION | MB_SETFOREGROUND);
        EndDialog(IDCANCEL);
    }
}

```

```

        return TRUE;
    }

    m_lstLabels.SetCurSel(0);
    m_iLabel = 0;

    return TRUE; // return TRUE unless you set the focus to a
control
                // EXCEPTION: OCX Property Pages should return
FALSE
}

void CBrowseDlg::OnOK()
{
    int iRowCount;

    UpdateData();

    // Get the IP Address corresponding to the selected list
item

    m_strIPAddress = "ows_IPAddress"; // To get the IP Address, we
must provide the field name
    iRowCount = CVideoPeer::camlQueryRowset(m_bstrXML,
m_strIPAddress, m_iLabel);
    if ((iRowCount < 1) || (m_strIPAddress.IsEmpty()) ||
(m_strIPAddress == "ows_IPAddress"))
    {
        m_strIPAddress.Empty();
        MessageBox("There was an error reading the IP address of
the selected label.\r\nPlease try again.", "Video Streaming System",
MB_ICONINFORMATION | MB_SETFOREGROUND);
        return;
    }

    // Close the dialog. The caller can obtain the IP Address
from the
    // public member variable m_strIPAddress

    EndDialog(IDOK);
}

void CBrowseDlg::lstLabels_OnDblClick()
{
    // A double-click on a list item is equivalent to pressing
"OK"

    OnOK();
}

bool CBrowseDlg::camlPopulateListWithRowset
(
    bstr_t      bstrXML,          // XML with rowset
    CListBox& lstPopulate        // Listbox to populate

```

```

    }
    {
        IXMLDOMDocument2Ptr spXML;
        IXMLDOMNodeListPtr spNodeList;
        IXMLDOMNodePtr      spNode;
        VARIANT_BOOL         vntbRes;
        int                  iRowCount;
        CString              strLabel;

        try
        {
            // Parse the XML response for the CAML data rows

            spXML.CreateInstance("Msxml2.DOMDocument.4.0");

            vntbRes = spXML->loadXML(bstrXML);
            if (vntbRes == VARIANT_FALSE) throw exception();
            spXML->setProperty("SelectionLanguage", "XPath");
            spXML->setProperty("SelectionNamespaces",
                "xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882' "
                "xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882' "
                "xmlns:rs='urn:schemas-microsoft-com:rowset' "
                "xmlns:z='#RowsetSchema'");
            spNodeList = spXML->selectNodes("//z:row");
            iRowCount = spNodeList->length;
            if (iRowCount < 1) return false;

            for (spNode = spNodeList->nextNode(); spNode != NULL;
                spNode = spNodeList->nextNode())
            {
                strLabel = (BSTR) spNode->attributes-
>getNamedItem(_bstr_t("ows_Title"))->text;
                lstPopulate.AddString(strLabel);
            }

            return true;
        }
        catch (...)
        {
            return false;
        }
    }
}

```

DEMANDES OU BREVETS VOLUMINEUX

**LA PRÉSENTE PARTIE DE CETTE DEMANDE OU CE BREVETS
COMPREND PLUS D'UN TOME.**

CECI EST LE TOME 2 DE 2

NOTE: Pour les tomes additionels, veuillez contacter le Bureau Canadien des Brevets.

JUMBO APPLICATIONS / PATENTS

**THIS SECTION OF THE APPLICATION / PATENT CONTAINS MORE
THAN ONE VOLUME.**

THIS IS VOLUME 2 OF 2

NOTE: For additional volumes please contact the Canadian Patent Office.
